ProofPower

HOL REFERENCE MANUAL

PPTex-2.9.1w2.rda.110727

Information on the current status of ProofPower is available on the World-Wide Web, at URL:

`http://www.lemma-one.demon.co.uk/ProofPower/index.html`

This document is published by:

Lemma 1 Ltd.
2nd Floor
31A Chain Street
Reading
Berkshire
UK
RG1 2HX
e-mail: pp@lemma-one.com

PPTex-2.9.1w2.rda.110727 13:54:16 27/07/2011

CONTENTS

ABOUT THIS PUBLICATION

0.1 Purpose

This document, one of several making up the user documentation for the ProofPower system, is the reference manual for the system.

0.2 Readership

This document is intended to be consulted by users already acquainted with the basic principles behind ProofPower who need detailed information on the behaviour of specific facilities provided by the system. It is not a tutorial for learning the basic use of the system. A 'keyword in context' index is supplied, which is useful for identifying the full range of facilities of a particular kind, provided that the reader is familiar with the naming conventions adopted in the development of ProofPower.

0.3 Related Publications

A bibliography is given at the end of this document. Publications relating specifically to ProofPower are:

1. ProofPower Tutorial *[4]*, tutorial covering the basic ProofPower system.
2. ProofPower Installation and Operation *[5]*;
3. ProofPower Document Preparation *[3]*.

0.4 Assumptions

It is assumed that the reader has some prior acqaintance with ProofPower either by attending a course on ProofPower or by reading the tutorial.

0.5 Acknowledgements

ICL gratefully acknowledges its debt to the many researchers (both academic and industrial) who have provided intellectual capital on which ICL has drawn in the development of ProofPower.

We are particularly indebted to Mike Gordon of The University of Cambridge, for his leading role in some of the research on which the development of ProofPower has built, and for his positive attitude towards industrial exploitation of his work.

The ProofPower system is a proof tool for Higher Order Logic which builds upon ideas arising from research carried out at the Universities of Cambridge and Edinburgh, and elsewhere. In particular the logic supported by the system is (at an abstract level) identical to that implemented in the Cambridge HOL system [1], and the paradigm adopted for implementation of proof support for the language follows that adopted by Cambridge HOL, originating with the LCF system developed at Edinburgh [2]. The functional language 'Standard ML' used both for the implementation and as an interactive metalanguage for proof development, originates in work at Edinburgh, and has been developed to its present state by an international group of academic and industrial researchers. The implementation of Standard ML on which ProofPower is based was itself originally implemented by David Matthews at the University of Cambridge, and is now commercially marketed by Abstract Hardware Limited.

UNIX INTERFACES

SML
hol_list [−*c*] [−*d database*[#*theoryname*]] [−*i scripts*] [−*v*] *theory* ...
hol_list [−*d database*[#*theoryname*]] [−*i scripts*] [−*v*]
hol_list [−*c*] [−*d database*] [−*i scripts*] [−*v*] −*a*

Description `hol_list` is used to obtain selected information from a ProofPower-HOL database. It functions in the same manner as `zed_list` except that it uses defaults appropriate to the ProofPower-HOL, and a HOL theory lister.

In the first form of use, where a list of one or more theory names is specified, `hol_list` uses ProofPower-HOL to generate on its standard output listings (in the HOL language using the function *output_theory*) of the indicated theories in a form suitable for processing by `doctex`. Any cache theory (i.e. the theory name is in the list returned by *get_cache_theories*) will be printed with most of the theory detail elided, unless the `-c` option is given.

In the second form, with no list of theory names, `hol_list` lists the names of all the theories in the database whose language is "HOL", in a sorted order, one per line on its standard output channel. The third form, with `-a`, is like the first but causes all of the theories in the database whose language is "HOL" to be listed in a sorted order.

In any of the three forms the program will start a session as if by command `hol` with the supplied `-d` and `-i` arguments (if any), and it is in this environment that the theory listing is done. The output of this startup will be suppressed, including any indication of failure to load the initialisation scripts.

Each theory is, if possible, made current, or at least in scope, when it is listed.

In any form `-v` indicates the log of the preprocessing should also be output.

Errors `hol_list` prints a message and exits (with the value 1) if the database or any of the theories does not exist. The log of the failure is sent to the standard output, the message to the error output.

See Also `pp_list`, `zed_list`, `pp`, `pp_make_database`

SML
hol [−*d database*[#*theoryname*]] [−*i files*] [−*f files* [−*n*|−*s*] [−*v*]] [−− *ml_flags*]
zed [−*d database*[#*theoryname*]] [−*i files*] [−*f files* [−*n*|−*s*] [−*v*]] [−− *ml_flags*]

Description `hol` and `zed` are identical to `pp`. q.v., except that they use default databases *hol* and *zed* respectively, and hence `-d database` is optional.

SML

pp_list $[-c]$ $-d$ *database*[#*theory*] $[-i$ *scripts*$]$ $[-l$ *lang*$]$ $[-v]$ *theory* ...
pp_list $-d$ *database*[#*theory*] $[-i$ *scripts*$]$ $[-l$ *lang1* $[-l$ *lang2* ...$]]$ $[-v]$
pp_list $[-c]$ $-d$ *database* $[-i$ *scripts*$]$ $[-l$ *lang1* $[-l$ *lang2* ...$]]$ $[-v]$ $-a$

Description `pp_list` is used to obtain selected information from a ProofPower database.

In the first form of use, where a list of one or more theory names is specified, `pp_list` uses ProofPower to generate on its standard output listings of the indicated theories held in the database given by the `-d` option in a form suitable for processing by `doctex`.

If there is no `-l` option then the theory lister used will depend on the language of the theory. If the language is "HOL" then *output_theory* is used. Otherwise it will attempt to use a function named:

<language in lower case>_output_theory

and only if that doesn't exist will it use *output_theory*. All but the first language will be ignored.

If the `-l lang` option is given then it will take the language code of all theories given to be `lang`, and then work as above.

If no `-d` option is given then the function fails.

Any cache theory (i.e. the theory name is in the list returned by *get_cache_theories*) will be printed with most of the theory detail elided, unless the `-c` option is given.

In the second form, with no list of theory names, `pp_list` lists the names of all the theories in the database one per line on its standard output channel in a sorted order. If any `-l` options are given then only theories whose language is one of those listed will be noted.

The third form, with `-a`, is like the first but causes all of the theories in the database to be listed in a sorted order. If any `-l` options are given then only theories whose language is one of those given will be listed, and they will be individually printed according to their own language.

In any of the three forms, the program will start a session as if by command `pp` with the supplied `-d` and `-i` arguments (if any), and it is in this environment that the theory listing is done. The output of this startup will be suppressed, including any indication of failure to load the initialisation scripts.

Each theory is, if possible, made current, or at least in scope, when it is listed.

In any form `-v` indicates the log of the preprocessing should also be output.

Errors `pp_list` prints a message and exits (with the value 1) if the database or any of the theories does not exist. The log of the failure is sent to the standard output, the message to the error output.

See Also `zed_list`, `hol_list`, `pp`, `pp_make_database`

SML

pp_make_database \
  $[-c][-v]$ $[-f]$ $[-p$ *parentdatabase*$[\#$*parenttheory*$]]$ *newdatabase*$[\#$*cachetheory*$]$

Description `pp_make_database` makes a new child database to contain ProofPower theories. The new database initially contains a single theory, called the *cache theory* for the database, with name given by `cachetheory` (which is used by certain system functions to cache various definitions and theorems and which is used as the initial current theory when the database is used by the `pp`, `hol` and `zed` commands). If `cachetheory` is omitted then the database name, prefixed by "cache' " is taken to be the same as the name of the new cache theory.

The `-p` option may be used to indicate the database which is to be the parent of the new database and to indicate which theory in it is to be the parent of the theory `cachetheory`. The parent theory is taken to be the cache theory for the parent database if it is not given explicitly.

For portability, the parent database name should normally be given without any architecture- or compiler-specific prefixes or suffixes. Any such prefixes or suffixes will be added automatically by `pp_make_database`, If the resulting file name is an absolute path name (i.e., starts with a '/' character), then that is used as the parent database file name. If the resulting file name is not an absolute path name, `pp_make_database` looks for the parent database file first in the current directory and then in the user's search path (given in the environment variable `$PATH`).

If the `-p` option is not supplied then the database `hol` supplied with the system is used as the parent database, and the parent theory is the theory *hol*. This is an appropriate default for a ProofPower-HOL child database. An appropriate value for ProofPower-Z might be the database `zed` supplied with the system.

In interactive use, `pp_make_database` will normally ask for confirmation before overwriting the database if it already exists. The `-f` (force) option may be used to suppress the request for confirmation before overwriting an existing database.

The `-v` option produces more output which may be useful for diagnostic purposes.

Under Poly/ML, databases are subject to an adjustable size limit. By default, `pp_make_database` will adjust the size limit of the parent database to the minimum possible and adjust the size limit of the child database to the maximum allowed. The `-c` option suppresses these adjustments.

The supplied child database name will be used to create the child database file name which is derived using an algorithm specific to the Standard ML compiler being used.

Errors `pp_make_database` prints a message and exits (with value 1) if the parent database or theory does not exist, if the new database cannot be created or if the name of the cache theory clashes with the name of a theory in the parent database.

Some systems impose a limit on the depth of nesting of the database hierarchy and the command will print an error message and exit (with value 1) if this limit would be exceeded.

The environment variable `PPCOMPILER` may be used to select between the Poly/ML or SML/NJ compiler if ProofPower has been installed for both compilers. If it is set, the value of this variable must be either "`POLYML`" or "`SMLNJ`".

See Also `hol`, `zed`, `pp`.

SML

pp $-d$ *database*[#*theoryname*] [$-i$ *files*] [$-f$ *files* [$-n$|$-s$] [$-v$]] [$--$ *ml_flags*]

Description pp runs ProofPower on the indicated database. If no `-d database` is provided to pp, the function fails. For portability, the database name should be given without any architecture- or compiler-specific prefixes or suffixes. Any such prefixes or suffixes will be added automatically by pp, If the resulting file name is an absolute path name (i.e., starts with a '/' character), then that is used as the database file name. If the resulting file name is not an absolute path name, pp searches for the database file using the search path given in the environment variable `$PPDATABASEPATH`, if set. If `$PPDATABASEPATH` is not set, pp searches for the database in the current directory, then in the subdirectory `db` of the user's home directory and then in the subdirectory `db` of the ProofPower installation directory.

If specified, *theoryname* gives the name of a theory to be made the current theory at the start of the session. If *theoryname* is not specified, then current theory will be set to the theory current when the database was last saved by *save_and_quit* or, if just created, to the cache theory for the database. The files identified by any `[-i files]` options are then executed in turn. *files* is a comma-separated list of files.

If `-f files` is provided, then the files specified in the list *files* are loaded in batch mode. Once loading is complete the database is saved and the batch session is terminated. The saving of the database can be suppress by providing the `-n` flag. The default action if any of the files fails to load is for the session to terminate at that point and the database is not saved. By providing the `-s` flag, the user can indicate to the system to save the database in batch mode upon failure. The `-n` and `-s` flags are mutually exclusive. If they are both provided, a warning message is issued and the `-s` flag is ignored.

By default, the production of subgoal package output in a batch load is as determined by the value of the flag *subgoal_package_quiet* stored in the database. If the `-v` flag is specified to pp, the subgoal package output is produced whereas if the `-q` flag is specified, it is suppressed.

If `-f files` is not provided, then the system then issues a prompt for user input.

Flags which appear after `--` are passed directly onto the Standard ML system for processing. This mechanism can be used to tailor the heap size under SML/NJ: e.g., `pp -d hol -- -h 32000`. The environment variable `PPCOMPILER` may be used to select between the Poly/ML or SML/NJ compiler if ProofPower has been installed for both compilers. If it is set, the value of this variable must be either "`POLYML`" or "`SMLNJ`".

The environment variable `PPLINELENGTH`, if set, determines the initial value of the string control *line_length*. This gives the line length used by various listing facilities, e.g., *print_theory* and *output_theory*. In interactive use, the `xpp` interface will set `PPLINELENGTH` automatically if it has not been set explicitly by the user.

Errors pp prints a message and exits (with status 1) if the database cannot be accessed or if the theory name specified as part of the `-d` argument does not exist in the database.

See Also `pp_make_database`, `pp_list`, `pp_read`, `hol`, `zed`

SML
xpp [*Standard X Toolkit options*] [*xpp options*]

Description The program xpp provides a convenient way to prepare, check and execute ProofPower scripts under the X Windows System. xpp combines a general purpose text editor with a command interface for operating the ProofPower specification and proof facilities. Consult the xpp help menu or the xpp *User Guide* for information on how to use it.

'Standard X toolkit options' refers to common options which are automatically supported by most X Windows applications. An example is the option '-display', which may be used to specify the X server on which you wish xpp output to be displayed.

The xpp option -f file may be used to specify a file to be loaded into the editor when xpp starts. If you omit this option, xpp will start off editing an empty file.

If you specify the xpp option -d database, xpp will run an interactive command session working on the specified ProofPower database. If you omit this option, xpp will just run as an editor.

The command line options mentioned above are the most common ones. The program has a number of other options you may wish to use. Consult the xpp *User Guide* for further details.

See Also *USR031: ProofPower - Xpp User Guide*

SML
zed_list [−*c*] [−*d database*[#*theory*]] [−*i scripts*] [−*v*] *theory ...*
zed_list [−*d database*[#*theory*]] [−*i scripts*] [−*v*]
zed_list [−*c*] [−*d database*] [−*i scripts*] [−*v*] −*a*

Description zed_list is used to obtain selected information from a ProofPower-Z database. It functions in the same manner as hol_list except that it uses defaults appropriate to the ProofPower-Z, and a Z theory lister.

In the first form of use, where a list of one or more theory names is specified, zed_list uses ProofPower-Z to generate on its standard output listings (in the Z language using the function *z_output_theory*) of the indicated theories, in a form suitable for processing by doctex. Any cache theory (i.e. the theory name is in the list returned by *get_cache_theories*) will be printed with most of the theory detail elided, unless the -c option is given.

In the second form, with no list of theory names, zed_list lists the names of all the theories whose language is Z in the database one per line on its standard output channel, in a sorted order.

The third form, with -a, is like the first but causes all of the theories in the database whose language is "Z" to be listed in a sorted order.

In any of the three forms the program will start a session as if by command zed with the supplied -d and -i arguments (if any), and it is in this environment that the theory listing is done. The output of this startup will be suppressed, including any indication of failure to load the initialisation scripts.

Each theory is, if possible, made current, or at least in scope, when it is listed.

In any form -v indicates the log of the preprocessing should also be output.

Errors zed_list prints a message and exits (with the value 1) if the database or any of the theories does not exist. The log of the failure is sent to the standard output, the message to the error output.

See Also pp_list, hol_list, zed, pp_make_database

SML

conv_ascii [$-r$] [$-K$] [$-k$ *keyword_file_name*] <*filename*> ...
conv_extended [$-r$] [$-K$] [$-k$ *keyword_file_name*] <*filename*> ...

Description *conv_ascii* converts ProofPower documents using the extended character set into ASCII keyword format. *conv_extended* performs the opposite conversion.

The *filename* arguments may be just the base-name, perhaps with a directory name prefix, or may include the `.doc` suffix. By default, the result of the conversion is checked by converting in the opposite direction and comparing with the input. If the check is successful, the `.doc` file is then replaced by the result of the conversion. If the conversion appears to be unsuccessful the output of the conversion is placed in a file with suffix `.asc` or `.ext` in the current directory, and the `.doc` file is left unchanged.

If $-r$ is specified no check is made and the output of the conversion is placed in a file with suffix `.asc` or `.ext`.

Note that the check will always fail on a file containing a mixture of extended characters and ASCII keywords. Use $-r$ and then, if all is well, overwrite the `.doc` file with the `.asc` or `.ext` file using *mv(1)* or *cp(1)* to convert a such file into a homogeneous one.

The check will also fail if the file is already in the desired format, in which case there is no need to run the conversion program.

The $-K$ and $-k$ options indicate the keyword files to be used as for *doctex* and *docsml* (and are only needed if fonts other than those supplied with ProofPower are being used.)

See Also *docpr*

SML

docdvi [$-v$] [$-f$ *view_file_name*] [$-K$] [$-k$ *keyword_file_name*]
[$-e$ *edit_file_name*] [$-p$ *TeX_program_name*] [$-\ \mathcal{N}$] <*filename*> ...

Description Shell script that combines the actions of `doctex`, `bibtex` (which is part of the basic TEX distribution) and `texdvi` with the intention of fully processing a simple document from its `.doc` form to a printable `.dvi` file.

The option $-\mathcal{N}$ controls how many times LATEX should be invoked, the default is three (i.e., '-3'), the values of $\mathcal{N}$ may be in the range one to four inclusive. The other options are as for `doctex` and `texdvi`.

LATEX and `bibtex` are run so that if they detect errors and prompt for input they will read an end of file and thus stop immediately.

In some cases an extra run of LATEX may be required. In these cases LATEX will output the message: '*LaTeX Warning: Label(s) may have changed. Rerun to get cross-references right.*'

See Also *doctex*, *texdvi*

SML

docpr [$-n$] [$-p$] [$-s$] [$-v$] [$-w$ *width*] <*filename*> ...

Description Shell script that prints out files that may contain extended characters in a verbatim-like manner. Lines may be numbered in the output by using the $-n$ option. Lines are folded at at 80 characters wide, or at the width given by the $-w$ *width* option. The output may be viewed on screen with the $-s$ option, the default is to print the output. By default all intermediate files are deleted, with the $-p$ option the `.dvi` file will be preserved. With the $-v$ option details of the files processed are listed on the standard output.

See Also *doctex*, *texdvi*

SML

doctex $[-v]$ $[-f\ \ view\_file\_name]$ $[-e\ \ edit\_script]$
$[-K]$ $[-k\ \ keyword\_file\_name]$ <*filename*> ...
docsml $[-v]$ $[-f\ \ view\_file\_name]$ $[-K]$ $[-k\ \ keyword\_file\_name]$ <*filename*> ...

Description Shell scripts that sieve each of their *filename* arguments to produce various output files. These arguments may be given just as the base-name, perhaps with a directory name prefix, or may include the `.doc` suffix. When the $-v$ option is set details of the files read and written are shown on the standard output. The default steering files are named `sieveview` and `sievekeyword` and looked for first in the current directory, second on the callers execution path (from the UNIX environment variable `$PATH`). The default viewfile may be changed with the $-f$ option. The default keyword file may be suppressed with the $-K$ option. Additional keyword files may be given with the $-k$ option which may be used several times. The $-e$ option identifies the name of a script of `ex` commands which are used to edit the `.tex` file.

The output file from `doctex` has suffix `.tex` and is intended for processing with `texdvi`. The output file from `docsml` has suffix `.sml` and is typically processed by loading it into a ProofPower database.

See Also *texdvi*, *docdvi*

SML

texdvi $[-v]$ $[-b]$ $[-p\ \ TeX\_program\_name]$ <*filename*> ...

Description Shell script that runs LaTeX on each of the *filename* arguments to produce the corresponding `.dvi` file. These arguments may be just the base-name, perhaps with a directory name prefix, or may include the `.tex` suffix. When the $-v$ option is set details of the `.tex` and`.dvi` files read and written are shown on the standard output. To support indexing this script ensures that a `.sid` file exists before LaTeX is called; when LaTeX completes any `.idx` file is sorted to create a`.sid` file ready for the next time `texdvi` is used. When initially producing a `.dvi` file `texdvi` will need to be run up to four times so that the derived information such as tables of contents and inter-page references stabilise.

The LaTeX program is `latex` by default but a different program may be specified with the $-p$ option.

If the `-b` option is specified, `bibtex` is run after running `latex`.

See Also *texdvi*, *docdvi*

PROGRAMMING UTILITIES

2.1 Error Management

SML
```
signature BasicError = sig
```

Description This is the signature of the structure *BasicError*.

SML
```
exception Fail of MESSAGE
exception Error of MESSAGE
```

Description These exception are raised to report error conditions. *Fail* is for errors which may be trapped (so that the associated message is suppressed). *Error* is intended to ensure that the message will be reported and, by convention, should not be trapped.

Uses Obscure debugging situations.

SML
```
type MESSAGE
```

Description This type is used to pass error and other messages around in the system.

Uses Obscure debugging situations.

SML
```
val area_of : exn -> string
```

Description This returns the name of the function which raised an exception (provided the exception was raised with *fail* following the usual conventions). If the exception was not the one raised by *fail* then it is raised again.

Uses For use when coding new facilities to add to the system.

SML
```
val divert : exn -> string -> string -> int -> (unit -> string) list -> 'a
val list_divert : exn -> string -> ((string * int * ((unit -> string) list)) list) -> 'a
val elaborate : exn -> int -> string -> int -> (unit -> string) list -> 'a
```

Description These functions support a style of error handling in which, if an error is reported during evaluation of an expression, the source of the error may be checked and the error report modified if needed to give a more meaningful report to the user. Sources of errors are identified by the string passed as the first argument to the function *fail* which is used to flag trappable errors. By convention, this string gives the name of the top level function which has raised the error.

In the call *divert X from new new_msg inserters*, *X* is the exception which has been raised and *from* identifies a possible source for an error report. *inserters* is a list of functions to be used to generate insertions for the error message (as with *fail* q.v.). If an error has been reported by *from*, the call will have the same effect as if *fail new new_msg inserters* had been called.

list_divert X new triples handles the more general case in which errors from several sources are expected. *X* and *new* are as for *divert*. *triples* gives a list of triples giving possible sources of error and the corresponding new messages and insertion functions.

elaborate is similar to *divert* but makes it possible to expand on the information provided by the function that has raised the exception. In the call *elaborate X old_msg new new_msg inserters*, *old_msg* identifies an error message text. If *X* results from a call of *fail* (or equivalent) with that error message text, the effect is as if *fail new new_msg (inserters'@inserters)* had been called, with *inserters'* the list of string-valued functions associated with *X*.

Uses For use when coding new facilities to add to the system.

SML
```
val fail : string -> int -> (unit -> string) list -> 'a
val error : string -> int -> (unit -> string) list -> 'a
```

Description These functions report a message of the corresponding class with text determined by an integer parameter and a list of string valued functions. The string parameter is intended to give the name of the top level function which has invoked the error message.

The error messages are stored in a database maintained by *new_error_message* and the integer parameter gives the key for the desired entry in the database. The list of string-valued functions allow the messages to be parameterised. When the error is printed, the functions are evaluated to produce a list of strings. Substrings of the database entry of the form "?*i*" where *i* is a decimal digit are replaced by the corresponding entries in the list (with "?*0*" corresponding to the head of the list). (If there are more than ten entries in the list, entries after the tenth are evaluated but the result of the evaluation is ignored).

fail is for unrecoverable errors which may, however, be trapped. It causes exception *Fail* to be raised.

error is for unrecoverable errors which must be reported to the user. It causes exception *Error* to be raised. As for *set_flag* etc.

Uses For use when coding new facilities to add to the system.

SML
```
val get_error_message : int -> (string list) -> string
```

Description This function returns the entry in the error message database associated with the given integer key. The second parameter gives a list of strings to be inserted into the text of the message. Substrings of the message text of the form "?*i*", where *i* is a decimal digit, indicate positions where these insertions are to be made. "?*0*" identifies the string at the head of the list etc.

Errors
```
2002 The error number ?0 does not identify an entry in the error message database
```

SML
```
val get_error_messages : unit -> {id:int, text:string} list
val set_error_messages : {id:int, text:string} list -> unit
```

Description *get_error_messages* returns the contents of the error message database as a list.

set_error_messages uses *new_error_message* to add any new error messages in a list of such into the database of error messages. It will issue a message on the standard output (and change nothing) for any messages that do not match those already present.

SML
```
val get_message_text: MESSAGE -> string
```

Description This returns a printable form of an error message text. The message text is given without the header information which is inserted by *get_message*, q.v.

Uses In constructing extensions to the system.

The error message data structure includes functions passed as arguments to *fail* or *error* that are called to generate parts of the message. If any of these functions raises *Fail*, the exception is caught and the string returned is a report on the failure.

See Also *fail*, *error*, *get_message*

Errors

2004 Failure detected formatting message: ?0
2005 ∗ failure ?0.?1 reported ∗

SML
```
val get_message: MESSAGE -> string
```

Description This returns a printable form of an error message value. The message text is followed by a trailer of the form "<*#nnnnn area*>", where *#nnnnn* is the number of the message in the error database and *area* typically gives the name of the function which gave rise to the error message.

Uses In constructing extensions to the system.

See Also *get_message_text*

SML
```
val new_error_message : {id:int, text:string} -> unit
```

Description This function adds a new entry to the database of error messages. Note that substrings of the message of the form "?*i*" where *i* is a decimal digit have special significance (see *fail* for details). "??" may be used to insert a single "?" character in a message.

If the *id* and the *text* are identical to an existing entry, then *new_error_message* has no effect. If there is an existing entry with the same *id* but a different *text* then a message is reported on the standard output and the existing entry is left unchanged.

Errors

2001 The error number ?0 is already in use for a different message

Uses For use when adding facilities to the system.

SML
```
val pass_on : exn -> string -> string -> 'a
```

Description *pass_on exn from to* is similar to *reraise*, q.v., but the function name associated with the exception is only modified if it is equal to *from*, in which case it is changed to *to*.

SML
```
val pending_reset_error_messages : unit -> unit -> unit
```

Description This function is intended for use in system initialisation and shutdown. The binding *val p = pending_reset_error_messages()*, defines *p* as a function which will set the internal state of the *BasicError* module to the value it had at the time the binding for *prcs* was made. This is used to remember the set-up for error messages introduced in a child database.

SML
|*val* **pp′change_error_message** : {*id*:*int*, *text*:*string*} −> *unit*

Description This function changes an entry in the database of error messages. If the number does not identify an existing entry a new entry is made.

Uses ICL Use only.

SML
|*val* **pp′error_init** : *unit* −> *unit*

Description This function is used to initialise certain aspects of the error reporting system. It is called automatically at the start of each session. It is harmless, but unnecessary, to call it within a session.

SML
|*val* **reraise** : *exn* −> *string* −> ′*a*

Description This re-raises an exception. If the exception is the exception *Fail* (as raised by *fail*, q.v.) then the function name associated with the exception is changed to the name given by the second argument.

Uses For use when coding new facilities to add to the system.

2.2 Data Types

SML
|*signature* **UtilitySharedTypes** = *sig*

Description Any new types in the Utility structures mentioned in more than one signature will be declared in this signature.

SML
|*datatype* ′*a* **OPT** = **Nil** | **Value** *of* ′*a*;

Description A type of "optional" values.

Uses A typical use for the datatype ′*a OPT* is in implementing partial functions for which raising an exception is not an appropriate action for undefined cases.

See Also *force_value*, *is_Nil*

SML
|*type* ′*a* **S_DICT**;

Description The type of simple dictionaries: (*string* ∗ ′*a*) *list*.

See Also Signature *SimpleDictionary*.

2.3 Lists

SML
```
signature ListUtilities = sig
```

Description Holds a variety of utility Standard ML list functions.

SML
```
val all_different : ''a list -> bool;
```

Description *all_different* determines whether a list has any repeated entries.

See Also *all_distinct*

SML
```
val all_distinct : ('a * 'a -> bool) -> 'a list -> bool;
```

Description *all_distinct eq list* determines whether *list* has any repeated entries using *eq* to test for equality. Each member, x of the list is tested against all the subsequent members of the list, with x being the first argument to *eq*.

See Also *all_different*

SML
```
val all : 'a list -> ('a -> bool) -> bool;
```

Description *all list cond* is true iff. all elements of *list* satisfy *cond*.

SML
```
val any : 'a list -> ('a -> bool) -> bool;
```

Description *any list cond* is true iff. some element of *list* satisfies *cond*.

SML
```
val app : ('a -> unit) -> 'a list -> unit;
```

Description Apply a function to each element of a list in turn for the side-effect.

SML
```
val combine : 'a list -> 'b list -> ('a * 'b) list;
```

Description *combine* combines a pair of lists into a list of pairs. It is the left inverse of *split*.

Errors
```
1007 Cannot combine unequal length lists
```

See Also *split*, *zip*

SML
```
val contains : ''a list -> ''a -> bool;
```

Description *contains list x* searches for a member of *list* equal to x and returns true iff. it finds one.

See Also *present*, *mem*

SML
```
val cup : ''a list * ''a list -> ''a list;
```

Description An infix binary union operation for lists, with Standard ML equality test. It has the same result ordering as *union*(q.v.).

See Also *list_cup*, *union*

SML
```
val diff : ''a list * ''a list -> ''a list;
```

Description *diff* is the set difference operator for lists.

SML
```
val drop : 'a list * ('a -> bool) -> 'a list;
```

Description *list drop cond* is the list obtained by deleting all members of *list* for which the boolean function *cond* is true.

See Also *less*

SML
```
val filter : ('a -> bool) -> 'a list -> 'a list;
```

Description *filter pred list* returns a list that is *list*, except that elements of the list that don't satisfy *pred* are dropped.

Definition
```
filter pred [] = []
| filter pred (a :: x) = (
        if pred a
        then (a :: filter pred x)
        else filter pred x);
```

SML
```
val find : 'a list -> ('a -> bool) -> 'a;
```

Description *find list cond* searches for the first member of *list* satisfying *cond*, and returns such a member if there is one.

Errors
```
1004    Element cannot be found in list
```

SML
```
val flat : 'a list list -> 'a list;
```

Description *flat* takes a list of lists and returns the result of concatenating them all.

SML
```
val fold : ('a * 'b -> 'b) -> 'a list -> 'b -> 'b;
```

Description Fold a list into a single value:

Definition
```
fold f [x1, x2, ...., xk] b = f(x1, f(x2, ... f (xk, b))...)
```

See Also revfold

SML
```
val force_value : 'a OPT -> 'a;
```

Description Force an object of type *'a OPT* (q.v) into one of type *'a*:

Definition
```
force_value (Value x) = x
```

Errors
```
1001    Argument may not be Nil
```

SML
```
val from : 'a list * int -> 'a list;
```

Description *list from n* takes the trailing slice of *list*. It uses 0-based indexing. If n is 0 or negative then entire list is returned, and if n indexes past the other end of the list then the empty list is returned.

Example
```
[0,1,2,3] from 2 = [2,3]
```

See Also *to*

SML
```
val grab : ''a list * ''a -> ''a list;
```

Description *list grab what* is the list obtained by inserting *what* at the head of *list* if it is not a member of it already, in which case *list* is returned.

See Also *insert*

SML
```
val hd : 'a list -> 'a;
val tl : 'a list -> 'a list;
```

Description *hd* returns first element of a list, *tl* returns all but the first element of a list.

Definition
```
hd (a :: x) = a
tl (a :: x) = x
```

Errors
```
1002    An empty list has no head
1003    An empty list has no tail
```

SML
```
val insert : ('a * 'a -> bool) -> 'a list -> 'a -> 'a list;
```

Description *insert eq list what*is the list obtained by inserting *what* at the head of *list* if it is not a member, by equality test *eq*, of it already, in which case *list* is returned.

See Also *grab*

SML
```
val interval : int -> int -> int list;
```

Description *interval a b* is the list $[a, a+1, a+2 \ldots, b]$. This is taken to be [] if $a > b$ and to be $[a]$ if $a = b$.

SML
```
val is_Nil : 'a OPT -> bool
```

Description Is the argument equal to *Nil* (q.v).

Definition
```
is_Nil Nil = true
| is_Nil _ = false
```

SML
```
val is_nil : 'a list -> bool;
```

Description *is_nil* tests whether a list is empty([]). It can be used for lists of types which do not admit equality.

SML
```
val lassoc1 : (''a * ''a) list -> ''a -> ''a;
```

Description *lassoc1 alist arg* is x, where $(arg,\ x)$ is the first element of *alist* with *arg* as its left item. The function is made total by taking *arg* as the result if there is no appropriate member of the list.

See Also lassoc$\mathcal{N}$ and rassoc$\mathcal{N}$, where $\mathcal{N} =\ 1 \ldots 5$.

SML
```
val lassoc2 : (''a * 'b) list -> (''a -> 'b) -> ''a -> 'b;
```

Description *lassoc2 alist f arg* is x, where $(arg,\ x)$ is the first element of *alist* with *arg* as its left item. The function is made total by returning *f arg* if there is no appropriate member of the list.

See Also lassoc$\mathcal{N}$ and rassoc$\mathcal{N}$, where $\mathcal{N} =\ 1 \ldots 5$.

SML
```
val lassoc3 : (''a * 'b) list -> ''a -> 'b;
```

Description *lassoc3 alist arg* is x, where $(arg,\ x)$ is the first element of *alist* with *arg* as its left item.

Errors
```
1005    No such value in association list
```

See Also lassoc$\mathcal{N}$ and rassoc$\mathcal{N}$, where $\mathcal{N} =\ 1 \ldots 5$.

SML
```
val lassoc4 : (''a * 'b) list -> 'b -> ''a -> 'b;
```

Description *lassoc4 alist default arg* is x, where $(arg,\ x)$ is the first element of *alist* with *arg* as its left item. The function is made total by returning *default* if there is no appropriate member of the list.

See Also lassoc$\mathcal{N}$ and rassoc$\mathcal{N}$, where $\mathcal{N} =\ 1 \ldots 5$.

SML
```
val lassoc5 : (''a * 'b) list -> ''a -> 'b OPT;
```

Description *lassoc5 alist arg* is *Value x*, where $(arg,\ x)$ is the first element of *alist* with *arg* as its left item. The function is made total by returning *Nil* if there is no appropriate member of the list.

See Also lassoc$\mathcal{N}$ and rassoc$\mathcal{N}$, where $\mathcal{N} =\ 1 \ldots 5$.

SML
```
val length : 'a list -> int;
```

Description *length* returns the length of a list. Note that the Standard ML function *size* can be used to find the length of strings.

SML
```
val less : ''a list * ''a -> ''a list;
```

Description *list less what* is the list obtained by deleting all members of *list* which are equal to *what*.

See Also *drop*

SML
```
val list_cup : ''a list list -> ''a list;
```

Description A distributed union operation for lists, with Standard ML equality test.

Definition
```
list_cup [list0, list1, ..., listn] =
        list0 cup (list1 cup ...(listn cup [])...))
```

See Also *cup*, *list_union*

SML
```
val list_overwrite : (''a * 'b) list * (''a * 'b) list -> (''a * 'b) list;
```

Description *alist list_overwrite olist* overwrites *alist* with each element of *olist*, using *overwrite*(q.v).

Definition
```
fun alist list_overwrite olist = (
        fold (fn (l1, l2) => l2 overwrite l1) olist alist
)
```

See Also *overwrite*, *list_roverwrite*.

SML
```
val list_roverwrite : ('a * ''b) list * ('a * ''b) list -> ('a * ''b) list;
```

Description *alist list_roverwrite olist* overwrites *alist* with each element of *olist*, using *roverwrite* (q.v.).

Definition
```
fun alist list_roverwrite olist = (
        fold (fn (l1, l2) => l2 roverwrite l1) olist alist
)
```

See Also *roverwrite*, *list_overwrite*.

SML
```
val list_union : ('a * 'a -> bool) -> 'a list list -> 'a list;
```

Description A distributed union operation for lists, with parameterised equality test:

Definition
```
list_union eq [list0, list1, ..., listn] =
        union eq list0 (union eq list1 (...(union eq listn [])...))
```

See Also *union*, *list_cup*.

SML
```
val mapfilter : ('a -> 'b) -> 'a list -> 'b list;
```

Description Map a function over a list. If, when evaluating

$$mapfilter\ f\ (x\_1 :: \ldots x\_k - 1 :: x\_k :: x\_k + 1 :: \ldots)$$

the evaluation of $f\ x\_k$ raises a *Fail* exception, then the result will be

$$(f\ x\_1 :: \ldots f\ x\_k - 1 :: f\ x\_k + 1 :: \ldots)$$

SML
```
val mem : ''a * ''a list -> bool;
```

Description *x mem list* searches for a member of *list* equal to *x* and returns true iff. it finds one.

See Also *contains*, *present*

SML
```
val nth : int -> 'a list -> 'a;
```

Description Return the n-th element of a list. The head of the list is the *0*-th element.

Errors
1009 Index past ends of list

SML
```
val overwrite : (''a * 'b) list * (''a * 'b) -> (''a * 'b) list;
```

Description *alist overwrite* (a, b) gives the list in which the first pair in *alist* that has the left item a is replaced with the pair (a, b). If no such pair is found in *alist* then it returns the list of (a, b) appended to the tail of *alist*.

See Also *roverwrite*, *list_overwrite*

SML
```
val present : (('a * 'a) -> bool) -> 'a -> 'a list -> bool;
```

Description *present eq x list* searches for a member, y, of *list* that satisfies $eq(x, y)$ and returns true iff. it finds one.

See Also *contains*, *mem*

SML
```
val rassoc1 : (''a * ''a) list -> ''a -> ''a;
```

Description *rassoc1 alist arg* is x, where $(x,\ arg)$ is the first element of *alist* with *arg* as its right item. The function is made total by taking *arg* as the result if there is no appropriate member of the list.

See Also lassoc$\mathcal{N}$ and rassoc$\mathcal{N}$, where $\mathcal{N} =$ *1 . . . 5*.

SML
```
val rassoc2 : ('a * ''b) list -> (''b -> 'a) -> ''b -> 'a;
```

Description *rassoc2 alist f arg* is x, where $(x,\ arg)$ is the first element of *alist* with *arg* as its left item. The function is made total by returning *f arg* if there is no appropriate member of the list.

See Also lassoc$\mathcal{N}$ and rassoc$\mathcal{N}$, where $\mathcal{N} =$ *1 . . . 5*.

SML
```
val rassoc3 : ('a * ''b) list -> ''b -> 'a;
```

Description *rassoc3 alist arg* is x, where $(x,\ arg)$ is the first element of *alist* with *arg* as its right item.

Errors
1005 No such value in association list

See Also lassoc$\mathcal{N}$ and rassoc$\mathcal{N}$, where $\mathcal{N} =$ *1 . . . 5*.

SML
```
val rassoc4 : ('a * ''b) list -> 'a -> ''b -> 'a;
```

Description *rassoc4 alist default arg* is x, where $(x,\ arg)$ is the first element of *alist* with *arg* as its right item. The function is made total by returning *default* if there is no appropriate member of the list.

See Also lassoc$\mathcal{N}$ and rassoc$\mathcal{N}$, where $\mathcal{N} =$ *1 . . . 5*.

SML
val **rassoc5** : $('a * ''b)\ list\ {-}{>}\ ''b\ {-}{>}\ 'a\ OPT$;

Description *rassoc5 alist arg* is *Value x*, where $(x,\ arg)$ is the first element of *alist* with *arg* as its right item. The function is made total by returning *Nil* if there is no appropriate member of the list.

See Also lassoc$\mathcal{N}$ and rassoc$\mathcal{N}$, where $\mathcal{N} =\ 1 \ldots 5$.

SML
val **revfold** : $('a * 'b\ {-}{>}\ 'b)\ {-}{>}\ 'a\ list\ {-}{>}\ 'b\ {-}{>}\ 'b$;

Description Fold a list into a single value:

Definition
$revfold\ f\ [x1,\ x2,\ ....,\ xk]\ b = f(xk,\ ...,\ f(x2,\ f\ (x1,\ b))...)$

See Also fold

SML
val **roverwrite** : $('a * ''b)\ list * ('a * ''b)\ {-}{>}\ ('a * ''b)\ list$;

Description *alist roverwrite* (a, b) gives the list that in which the first pair in *alist* that has the right item b is replaced with the pair (a, b). If no such pair is found in *alist* then it returns the list of (a, b) appended to the end of *alist*.

See Also *overwrite*, *list_roverwrite*

SML
val **split3** : $('a * 'b * 'c)\ list\ {-}{>}\ 'a\ list * 'b\ list * 'c\ list$;

Description Split a list of triples into a triple of lists. *split3* is the analogue of *split* for lists of triples.

See Also *split*

SML
val **split** : $('a * 'b)\ list\ {-}{>}\ 'a\ list * 'b\ list$;

Description Split a list of pairs into a pair of lists.

Definition
$split\ [(x0,\ y0),\ (x1,\ y1),\ ...\ (xk,\ yk)] = [x0,\ x1,\ ...\ ,\ xk],\ \ [y0,\ y1,\ ...\ ,\ yk]$

See Also *split3*, *combine*

SML
val **subset** : $''a\ list * ''a\ list\ {-}{>}\ bool$;

Description *l1 subset l2* is true iff. all the elements of *l1* are also elements of *l2*

See Also =

SML
val **to** : $'a\ list * int\ {-}{>}\ 'a\ list$;

Description *list to n* takes the initial slice of *list*. It uses 0-based indexing. If n is 0 or negative an empty list is returned, and if n indexes past the other end of *list* then the entire list is returned.

Example
$[0,1,2,3]\ to\ 2 = [0,1,2]$

See Also *from*

SML

val **union** : $('a * 'a \text{ -> } bool) \text{ -> } 'a\ list \text{ -> } 'a\ list \text{ -> } 'a\ list;$

Synopsis A prefix binary union operation for lists, with parameterised equality test.

Description *union* is essentially a binary union operation for lists. Since we need it to work on types which are not equality types, it has a parameter giving the relation to be used to determine equality of members of the lists. In some cases it may be important for the order of members of the union to be known. The rule is that *union eq list1 list2* is the list obtained by prepending those elements of *list1* not already present in *list2*, to the list *list2*. Presence for x in the list being created being that there is a member, y, of the list being created with $eq(x, y) = true$. If *list1* contains duplicates then all but the rightmost will be eliminated, but those in *list2* will not be. Note also that if one of the lists is small it is better supplied as the first list argument if efficiency is of the essence.

Definition

```
union eq (list1 @ [a]) list2 = union eq list1 (
        if present eq a list2
        then list2
        else (a :: list2)
) | union eq [] list2 = list2
```

See Also *cup*, *list_union*

SML

val **which** : $(('a * 'a) \text{ -> } bool) \text{ -> } 'a \text{ -> } 'a\ list \text{ -> } int\ OPT;$

Description *which eq x list* returns *Value* of the position of first element, y, in *list* for which *eq x y* is true. It uses 0-based indexing. If no such y is found, then it returns *Nil*.

SML

val **zip** : $('a \text{ -> } 'b)list \text{ -> } 'a\ \ list \text{ -> } 'b\ list;$

Description Given a list of functions, and a list of arguments, of the same length, apply each function to its corresponding argument. For the cases when the list of functions induce side effects, note that the functions are applied from the head of their list to the tail, and will be applied until there are insufficient elements of either list to continue. If there lists are not of equal length then at that point a failure will be raised.

See Also *combine*

Errors

1008 List lengths differ

SML

$val \sim\text{<=} : ''a\ list * ''a\ list \text{ -> } bool;$
$val \sim\text{=} : ''a\ list * ''a\ list \text{ -> } bool;$

Description *l1* <= *l2* is true iff. every member of *l1* is also a member of *l2*. *l1* = *l2* is true iff. the set of members of *l1* is equal to the set of members of *l2*.

See Also *subset*

2.4 Functions

SML
```
signature FunctionUtilities = sig
```

Description Holds a variety of utility Standard ML functions concerned with functions.

SML
```
val ** : ('a -> 'b) * ('c -> 'd) -> 'a * 'c -> 'b * 'd;
```

Description The infix operator $**$, with precedence 4 (higher than "o"), applies the first of a pair of functions to the first of a pair, and the second of the pair of functions to the second of the pair, returning the pairing of the results.

Definition
```
(f ** g) x = (f x, g x)
```

SML
```
val curry : ('a * 'b -> 'c) -> 'a -> 'b -> 'c;
```

Description *curry f a b* gives $f\ (a, b)$.

See Also *uncurry*

SML
```
val fst : 'a * 'b -> 'a;
```

Description *fst* is the left projection function for pairs: $fst(a, b) = a$.

See Also *snd*

SML
```
val fun_and : (('a -> bool) * ('a -> bool)) -> 'a -> bool;
val fun_or : (('a -> bool) * ('a -> bool)) -> 'a -> bool;
val fun_not : ('a -> bool) -> 'a -> bool;
val fun_true : 'a -> bool;
val fun_false : 'a -> bool;
```

Description These functions allow a style of programming that handles predicates rather than booleans.

Definition
```
(f fun_and g) x    = f x andalso g x
(f fun_or g) x     = f x orelse g x
(fun_not f) x = not(f x)
fun_true x    = true
fun_false x   = false
```

SML
```
val fun_pow : int -> ('a -> 'a) -> 'a -> 'a;
```

Description For non-negative n, *fun_pow n f* is f^n, i.e. the function

$$\lambda x \bullet f(f(...f(fx)....))$$

where f appears n times.

Errors
```
1010 First argument must not be negative
```

SML

val **repeat** : $(unit \mathrel{->} {'a}) \mathrel{->} unit;$
val **iterate** : $({'a} \mathrel{->} {'a}) \mathrel{->} {'a} \mathrel{->} {'a};$

Description *repeat* applies its argument to () until it fails (with an error generated by *fail*, q.v.), whereupon it returns (). *iterate f a* applies f to a. If this causes no failure it then calls *iterate f* on the result. If it fails (with an error generated by *fail*, q.v.) it returns a. Failures other than those caused by *fail* are not handled.

Definition

$fun\ repeat\ f \quad = (f\ (); repeat\ f)\ handle\ (Fail\ \_) => ()$
$fun\ iterate\ f\ a \quad = (iterate\ f\ (f\ a))\ handle\ (Fail\ \_) => a$

SML

val **snd** : $'a * 'b \mathrel{->} 'b;$

Description *snd* is the right projection function for pairs: $snd(a, b) \ = \ b$.

See Also *fst*

SML

val **swap** : $'a * 'b \mathrel{->} 'b * 'a;$

Description *swap* interchanges the elements of a pair: $swap(a, b) \ = \ (b, a)$.

SML

val **switch** : $('a \mathrel{->} 'b \mathrel{->} 'c) \mathrel{->} 'b \mathrel{->} 'a \mathrel{->} 'c;$

Description *switch f a b* gives *f b a*.

SML

val **uncurry** : $('a \mathrel{->} 'b \mathrel{->} 'c) \mathrel{->} 'a * 'b \mathrel{->} 'c;$

Description *uncurry f* (a, b) gives *f a b*.

See Also *curry*

2.5 Combinators

SML

signature **Combinators** = *sig*

Description Holds the three combinators S, K, I.

SML

val **I** : $'a\ {-}{>}\ 'a$

Description The identity combinator: $I\ x\ =\ x$.

SML

val **K** : $'a\ {-}{>}\ 'b\ {-}{>}\ 'a$

Description The deletion combinator: $K\ x\ y$ is x.

SML

val **S** : $('a\ {-}{>}\ 'b\ {-}{>}\ 'c)\ {-}{>}\ ('a\ {-}{>}\ 'b)\ {-}{>}\ 'a\ {-}{>}\ 'c$

Description The duplication combinator: $S\ f\ g\ a$ is $(f\ a)(g\ a)$.

2.6 Characters

SML
```
signature CharacterUtilities = sig
```

Description Holds a variety of utility Standard ML functions concerned with character handling.

SML
```
val is_all_decimal : string -> bool;
```

Description *is_all_decimal* checks whether a string consists of one or more decimal digits.

SML
```
val nat_of_string : string -> int;
```

Description *nat_of_string* converts a string into non-negative integer (using decimal notation).

See Also *string_of_int*

Errors
```
1012 ?0 is not a decimal string
1013 String is empty
```

SML
```
val string_of_int : int -> string;
```

Description *string_of_int* converts an integer into a decimal string.

See Also *nat_of_string*

2.7 Simple Dictionary

SML
signature **SimpleDictionary** = *sig*

Description Holds a set of Standard ML functions concerned with a linear search dictionary.

Uses For handling small dictionaries.

See Also *EfficientDictionary*.

SML
val **initial_s_dict** : *'a S_DICT*;

Description The empty dictionary, which gives a starting point for the use of the simple dictionary functions. It does not associate a value with any name.

SML
val **s_delete** : *string −> 'a S_DICT −> 'a S_DICT*;

Description *s_delete* deletes an element of the domain of a dictionary. If the element is not in the domain it returns the dictionary unchanged. *s_delete name dict* returns a dictionary that does not associate anything with *name*, but otherwise associates as *dict*.

SML
val **s_enter** : *string −> 'a −> 'a S_DICT −> 'a S_DICT*;

Description *s_enter* implements overwriting by a singleton function. *s_enter name value dict* returns the dictionary that associates *name* with *value*, and otherwise associates as *dict*. Overwriting is done "in place", entries not previously present will be placed at the end of the dictionary viewed as a list.

SML
val **s_extend** : *string −> 'a −> 'a S_DICT −> 'a S_DICT*;

Description *s_extend* implements extension by a singleton function, that is to say it is like *s_enter*. *s_extend name value dict* returns the dictionary that associates *name* with *value*, and otherwise associates as *dict*. It fails if *name* is already in the domain of *dict*. Entries not previously present will be placed at the head of the dictionary viewed as a list.

Errors
1014 ?0 is already in dictionary

SML
val **s_lookup** : *string −> 'a S_DICT −> 'a OPT*;

Description *s_lookup* implements application (of the dictionary viewed as a partial function). *s_lookup name dict* returns the value that *dict* associates with *name*.

SML
val **s_merge** : *'a S_DICT −> 'a S_DICT −> 'a S_DICT*;

Description *s_merge* extends one dictionary by another. The dictionary *s_merge dict1 dict2* will associate a name with the value that either *dict1* or *dict2* associates it with.

Failure Will get the *s_extend* failure message if any element is common to the domains of both dictionaries (*dict1* and *dict2*). Duplicate keys in the first list will also cause an *s_extend* error, but will be replicated in the result if found in the second list.

2.8 Efficient Dictionary

SML
```
signature EfficientDictionary = sig
```

Description This is the signature of a structure implementing dictionaries (lookup-up tables) based on hash-search techniques.

Uses For handling large dictionaries.

See Also *SimpleDictionary*.

SML
```
type 'a E_DICT;
```

Description The type of efficient dictionaries.

SML
```
type E_KEY;
val e_get_key : string -> E_KEY;
val e_key_lookup : E_KEY -> 'a E_DICT -> 'a OPT
val e_key_enter : E_KEY -> 'a -> 'a E_DICT -> 'a E_DICT;
val e_key_extend : E_KEY -> 'a -> 'a E_DICT -> 'a E_DICT;
val e_key_delete : E_KEY -> 'a E_DICT -> 'a E_DICT;
val string_of_e_key : E_KEY -> string;
```

Description The abstract data type *E_KEY* represents the hash-keys used in the internals of the efficient dictionary (*E_DICT*) access functions. *e_get_key* computes the hash-key for a given string. This may then be used as an argument to the functions *e_key_lookup*, *e_key_enter*, *e_key_extend* and *e_key_delete* which perform the same functions as the corresponding functions without "*key_*" in the name. This approach may be used if the same string is to be used to access several efficient dictionaries to avoid the computational cost of recalculating the hash-key. *string_of_key* is the left inverse of *e_get_key*.

Failure The failures are exactly as for the corresponding string access functions. In particular, the area names in error messages are, e.g., "e_lookup" rather than "e_key_lookup" etc.

SML
```
val e_delete : string -> 'a E_DICT -> 'a E_DICT;
```

Description *e_delete* deletes an element of the domain of a dictionary. If the element is not in the domain it returns the dictionary unchanged. *e_delete name dict* returns a dictionary that does not associate anything with *name*, but otherwise associates as *dict*.

SML
```
val e_enter : string -> 'a -> 'a E_DICT -> 'a E_DICT;
```

Description *e_enter* implements overwriting by a singleton function. *e_enter name value dict* returns the dictionary that associates *name* with *value*, and otherwise associates as *dict*.

SML
```
val e_extend : string -> 'a -> 'a E_DICT -> 'a E_DICT;
```

Description *e_extend* implements extension by a singleton function, that is to say it is like *e_enter*. *e_extend name value dict* returns the dictionary that associates *name* with *value*, and otherwise associates as *dict*. It fails if *name* is already in the domain of *dict*.

Errors

1014 ?0 is already in dictionary

SML

```
val e_flatten : 'a E_DICT -> 'a S_DICT;
```

Description *e_flatten* converts an efficient dictionary into a simple one. The result will contain no duplicates, but will be in no useful order.

SML

```
val e_lookup : string -> 'a E_DICT -> 'a OPT
```

Description *e_lookup* implements application (of the dictionary viewed as a partial function). *e_lookup name dict* returns the value that *dict* associates with *name*.

SML

```
val e_merge : 'a E_DICT -> 'a E_DICT -> 'a E_DICT;
```

Description *e_merge* extends one efficient dictionary by another. The dictionary *e_merge dict1 dict2* will associate a name with the value that either *dict1* or *dict2* associates it with.

Failure Will get the *e_extend* failure message if an element is common to the domains of both dictionaries.

SML

```
val e_stats : 'a E_DICT -> {height : int, nentries : int, nnodes : int, sumweights : int};
```

Description *e_stats dict* returns statistics about the internals of the efficient dictionary *dict*. Efficient dictionaries are currently represented as binary trees whose non-leaf nodes each carry a simple dictionary of entries (in case of collisiion of hash values). The statistics currently returned are the height of the tree, the number of entries, the number of nodes and the sum over all entries of the depth of the entries (i.e, the sum of the number of entries per node weighted by node-depth).

SML

```
val initial_e_dict : 'a E_DICT;
```

Description The empty dictionary, which gives a starting point for the use of the efficient dictionary functions. It does not associate a value with any name.

SML

```
val list_e_enter : 'a E_DICT -> 'a S_DICT -> 'a E_DICT;
```

Description *list_e_merge* extends an efficient dictionary by overwriting with entries from a simple one. That is, for each association in the simple dictionary an *e_enter* is executed on the efficient dictionary.

SML

```
val list_e_merge : 'a E_DICT -> 'a S_DICT -> 'a E_DICT;
```

Description *list_e_merge* extends an efficient dictionary by merging with entries from a simple one. That is, for each association in the simple dictionary an *e_extend* is executed on the efficient dictionary.

Failure Will get the *e_extend* failure message if an element is common to the domains of both dictionaries.

2.9 Order-preserving Efficient Dictionary

SML

```
type 'a OE_DICT;
val initial_oe_dict : 'a OE_DICT;
val oe_lookup : string -> 'a OE_DICT -> 'a OPT
val oe_enter : string -> 'a -> 'a OE_DICT -> 'a OE_DICT;
val oe_extend : string -> 'a -> 'a OE_DICT -> 'a OE_DICT;
val oe_delete : string -> 'a OE_DICT -> 'a OE_DICT;
val oe_key_lookup : E_KEY -> 'a OE_DICT -> 'a OPT
val oe_key_enter : E_KEY -> 'a -> 'a OE_DICT -> 'a OE_DICT;
val oe_key_extend : E_KEY -> 'a -> 'a OE_DICT -> 'a OE_DICT;
val oe_key_delete : E_KEY -> 'a OE_DICT -> 'a OE_DICT;
val oe_flatten : 'a OE_DICT -> 'a S_DICT;
val oe_key_flatten : 'a OE_DICT -> ('a * E_KEY) S_DICT;
val e_dict_of_oe_dict : 'a OE_DICT -> 'a E_DICT;
val list_oe_merge : 'a OE_DICT -> 'a S_DICT -> 'a OE_DICT;
val oe_merge : 'a OE_DICT -> 'a OE_DICT -> 'a OE_DICT;
```

Description This type and associated access functions implement order-preserving efficient dictionaries. The functions have exactly the same effect as the corresponding functions *e_lookup*, *e_enter* etc., qv., for the type *E_DICT* except that *se_flatten* returns a list which preserves the order in which entries were made (last-in, first-out). If an entry is updated (rather than added) by *oe_key_enter* or *oe_enter*, the updated entry appears in its original position.

list_oe_merge enters the list of items in the second argument into the dictionary given as its first argument in tail-first (right-to-left) order.

Failure The failures are exactly as for the corresponding *E_DICT* functions. In particular, the area names in error messages are, e.g., "e_lookup" rather than "oe_lookup" etc.

2.10 Orderings

SML
$type\ 'a$ **ORDER** $= 'a\ {-}{>}\ 'a\ {-}{>}\ int;$

Description This is a polymorphic type used by several functions (e.g., *sort*) to represent various kinds of ordering, e.g., linear orders or quasi-orders.

If f is an ordering in this sense, then elements x and y are taken to be in order ($x \prec y$), equivalent ($x \simeq y$) or out of order ($x \succ y$) according as $f\ x\ y$ is negative, zero or positive. So, for example, the usual ordering of the integers corresponds to $f\ x\ y = x - y$.

SML
val **ascii_order** : *string ORDER*;

Description This function gives the ordering of character string by numeric character code. The integer result is appropriate for use in *sort* (q.v.).

See Also *strin_order* which is a slightly less efficient but order alphabetic characters in a more intuitive way.

SML
val **induced_order** : $('a\ {-}{>}\ 'b) * 'b\ ORDER\ {-}{>}\ 'a\ ORDER;$

Synopsis Given an ordering r on elements of type $'b$ and a function f of type $'a\ {-}{>}'\ b$, $induced\_order(f, r)$ is the ordering of the type $'a$ induced by f and r, i.e., is is the ordering obtained by mapping elements of type $'a$ to the type $'b$ using f and then comparing the results with r.

SML
val **int_order** : *int ORDER*;

Description This function gives the usual ordering on the standard ML integers.

SML
val **lexicographic** : $'a\ ORDER\ {-}{>}\ 'a\ list\ ORDER;$

Synopsis This function returns the lexicographic ordering function induced from an ordering of items of type $'a$ on items of type $'a\ list$.

Description Lexicographic ordering may be defined as follows:

Let us presume some ordering < on the type of variables a_i and b_i. Then

$$[a\_1, \ldots, a\_n]\ <\_lexicographic\ [b\_1, \ldots, b\_m]$$

If, and only if, either

$$n\ <\_int\ m \land \forall i : 1..n \bullet a\_i\ =\ b\_i$$

Or

$$\exists j : 1..n \bullet (\forall i : 1..(j-1) \bullet a\_i\ =\ b\_i) \land (a\_j\ <\ b\_j)$$

The resulting ordering is appropriate for use in *sort* (q.v.).

list_order is a synonym for *lexicographic*.

See Also *string_order*, *pair_order*, *sort*, *list_order*.

SML
val **no_order** : $'a\ ORDER;$

Description This function is the ordering which makes any two elements compare equivalent.

SML
```
val pair_order : 'a ORDER -> 'b ORDER -> ('a * 'b) ORDER;
val list_order : 'a ORDER -> 'a list ORDER;
val inv_list_order : 'a ORDER -> 'a list ORDER;
val opt_order : 'a ORDER -> 'a OPT ORDER;
val opt_order1 : 'a ORDER -> 'a OPT ORDER;
val rev_order : 'a ORDER -> 'a ORDER;
```

Synopsis Several functions such as *sort* are parameterised by ordering functions, f, say, such that, $f\ x\ y$ is negative, zero or positive result according as $x < y$, $x = y$ or $x > y$ with respect to some total ordering. The above functions help to construct new ordering functions from old.

Given orderings on the component types of a binary product type, *pair_order* gives the ordering on the product type that orders first on the left component and then on the right to break ties.

Given an ordering on the element type of a list type, *list_order* gives the lexicographic ordering on the list type. It is a synonym for *lexicographic*, q.v.

inv_list_order is like *list_order* but gives the inverse lexicographic order (equivalent to reversing both operand lists but more efficiently implemented).

Given an ordering on any type, *opt_order* (resp. *opt_order1* gives an ordering on the type of optional values of that type. *Nil* is taken as greater (resp. less) than any supplied value, *Value x*.

Given an ordering on any type, *rev_order* gives the corresponding reverse ordering.

See Also *string_order*, *lexicographic*, *sort*

SML
```
val string_order : string ORDER;
```

Description This function gives a more civilised ordering on strings than the one given by the ASCII codes. It collates upper-case and lower-case letters together in the (increasing) order 'A', 'a', . . . , 'Z', 'z' and all other characters after 'z' ordered by numeric character code. The integer result is appropriate for use in *sort* (q.v.).

See Also *lexicographic*, *ascii_order*.

SML
```
val THEN_O : 'a ORDER * 'a ORDER -> 'a ORDER;
```

Synopsis *THEN_O* is an infix operator of precedence 4. Given orderings r and s on elements of type $'a$, r *THEN_O* s is the ordering that first compares elements using r and accepts the comparison unless the elements compare equivalent, in which case they are compared using s. This is equivalent to $induced\_order(diag, pair\_order\ r\ s)$ where $diag\ x = (x, x)$.

2.11 Sorting

SML

signature **Sort** = *sig*

include Order;

Description This provides an efficient sort utility package. For historical reasons it includes the structure *Order*.

SML

val **sort** : *'a ORDER -> 'a list -> 'a list*
val **merge** : *'a ORDER -> 'a list -> 'a list -> 'a list*

Description *sort* sorts a list and *merge* merges two lists assumed already to be sorted. Both functions are parametrised by an ordering function of type *'a ORDER*, i.e., $'a->' a-> int$. The integer, say n returned by an application of this function, say $f\ a\_1\ a\_2$, is interpreted as follows:

$n < 0$ *a_2* is to come after *a_1* (i.e. the arguments are in order).

$n > 0$ *a_2* is to come before *a_1* (i.e. the arguments are out of order).

$n = 0$ *a_2* is to be taken as equal to *a_1*

Sorting eliminates duplicate elements in the sense of the equality test given by the ordering. Merging includes just one copy of an element that occurs once in each of its arguments in the result. The result of merging unsorted lists is unspecified; in particular, the result is unspecified if there is duplication within one of the lists.

Example

To sort a list of integers, ilist in ascending order:

```
sort (curry (op -)) ilist
```

or

```
sort int_order ilist
```

See Also For convenient ways of constructing orderings, see, e.g. *string_order* and *list_order*.

2.12 Sparse Arrays

SML
signature **SparseArray** = *sig*

Description This is the signature of a structure implementing sparse arrays (i.e. imperative data structures representing finite partial functions on the integers). The sparse arrays also give an efficient means for handling dense (i.e. contiguous) arrays whose size varies. To facilitate their use for such dynamically sized arrays, the sparse arrays have lower and upper bound attributes which gives the smallest and largest indices into the array which identify an occupied cell.

The design of the structure is an adaptation of the library structure *Array* implementing fixed length arrays.

SML
type *'_a* **SPARSE_ARRAY**;

Description This is the type of a sparse array with entries of type *'_a*.

SML
val **array** : *int* −> *'_a SPARSE_ARRAY*;

Description This function creates an empty sparse array. The parameter indicates the length of an internal data structure used to represent the array. For a contiguous array or for a sparsely filled array with a random distribution of occupied cells, the average access time for an element will be proportional to n/l where n is the number of occupied cells and l is this length.

Errors
1102 The length parameter must be positive

SML
val **lindex** : *'_a SPARSE_ARRAY* −> *int*
val **uindex** : *'_a SPARSE_ARRAY* −> *int*

Description *lbound(array)* (resp. *ubound(array)*) returns the smallest (resp. largest) index of an occupied cell in the sparse array *array*. An exception is raised if the array is empty.

Errors
1103 the array is empty

SML
val **scratch** : *'_a SPARSE_ARRAY* −> *unit*;

Description *scratch array* empties all cells in the sparse array *array*.

SML
val **sub_opt** : (*'_a SPARSE_ARRAY* ∗ *int*) −> *'_a OPT*

Description *sub(array, i)* returns *Value a*, where a is the occupant of the i-th cell of the sparse array *array*. If the cell is unoccupied it returns *Nil*.

SML
val **sub** : (*'_a SPARSE_ARRAY* ∗ *int*) −> *'_a*

Description *sub(array, i)* returns the occupant of the i-th cell of the sparse array *array*. An exception is raised if the cell is not occupied.

Errors
1101 Cell with index ?0 is empty

SML

val **update** : $('\_a\ SPARSE\_ARRAY * int * '\_a) -> unit;$

Description $update(array, i, a)$ makes a the occupant of the i-th cell of the sparse array $array$. The cell need not previously have been occupied (indeed, *update* is the only means by which cells become occupied).

2.13 Dynamic Arrays

SML
signature **DynamicArray** = *sig*

Description This is the signature of a structure implementing dynamic arrays with 0-based indexing (i.e. imperative data structures representing finite partial functions on the integers, whose range is an interval *1 . . . n*). The implementation gives constant access time. The design of the structure is an adaptation of the library structure *Array* implementing fixed length arrays.

SML
type '_a **DYNAMIC_ARRAY**;

Description This is the type of a dynamic array with entries of type *'_a*.

SML
val **array** : *int -> '_a DYNAMIC_ARRAY*;

Description This function creates an empty dynamic array. The parameter indicates the length of an internal data structure used to represent the initial size of the array. The average access time for an element will be constant — the underlying array structure is grown as necessary.

Errors
1301 The initial size parameter must be positive

SML
val **scratch** : *'_a DYNAMIC_ARRAY -> unit*;

Description *scratch array* empties all cells in the sparse array *array* and reduces the underlying data structure to the initial length specified when the array was first created.

SML
val **sub_opt** : *('_a DYNAMIC_ARRAY * int) -> '_a OPT*

Description *sub(array, i)* returns *Value a*, where *a* is the occupant of the *i*-th cell of the dynamic array *array*. If the cell is unoccupied it returns *Nil*.

SML
val **sub** : *('_a DYNAMIC_ARRAY * int) -> '_a*

Description *sub(array, i)* returns the occupant of the *i*-th cell of the dynamic array *array*. An exception is raised if the cell is not occupied.

Errors
1101 Cell with index ?0 is empty
1303 Index ?0 is out of range

SML
val **uindex** : *'_a DYNAMIC_ARRAY -> int*

Description *lbound(array)* the largest index of an occupied cell in the dynamic array *array* or *~1* if no cells in the array are occupied.

SML
val **update** : *('_a DYNAMIC_ARRAY * int * '_a) -> unit*;

Description *update(array, i, a)* makes *a* the occupant of the *i*-th cell of the sparse array *array*. The cell need not previously have been occupied (indeed, *update* is the only means by which cells become occupied).

2.14 Arbitrary Magnitude Integer Arithmetic

SML

```
signature Integer = sig
    eqtype INTEGER;
    val idiv : INTEGER * INTEGER -> INTEGER;
    val imod : INTEGER * INTEGER -> INTEGER;
    val @* : INTEGER * INTEGER -> INTEGER;
    val @+ : INTEGER * INTEGER -> INTEGER;
    val @- : INTEGER * INTEGER -> INTEGER;
    val @~ : INTEGER -> INTEGER;
    val iabs : INTEGER -> INTEGER;
    val @< : INTEGER * INTEGER -> bool;
    val @> : INTEGER * INTEGER -> bool;
    val @<= : INTEGER * INTEGER -> bool;
    val @>= : INTEGER * INTEGER -> bool;
    val integer_of_string : string -> INTEGER;
    val @@ : string -> INTEGER;
    val string_of_integer : INTEGER -> string;
    val int_of_integer : INTEGER -> int;
    val integer_of_int : int -> INTEGER;
    val natural_of_string : string -> INTEGER;
    val zero : INTEGER;
    val one : INTEGER;
    val string_of_float : INTEGER * INTEGER * INTEGER -> string;
    val integer_order : INTEGER -> INTEGER -> int;
```

Description This is the signature of an open structure providing arithmetic on integers of arbitrary magnitude. It is used to support HOL natural numbers and other object language numeric types. The names of the usual arithmetic operators are decorated with an initial i or @ as appropriate. The string conversions work with signed decimal string representations. Either '−' or '∼' may be used for unary negation and a leading '+ ' is also allowed. @@ is an abbreviation for *integer_of_string*. *natural_of_string* is a converter for non-negative numbers (it has the same error cases as *nat_of_string*).

string_of_float interprets a triple (x, p, e) as a floating point number with value $x \times 10^{e-p}$ and converts the triple into its string representation.

integer_order implements the ordering of the integers in the form used by *sort*, q.v.

Errors

1201 the divisor is zero
1202 an empty string is not a valid decimal number
1203 the string '?0' is not a valid decimal number
1204 the conversion would overflow

2.15 Compatibility with SML'90

SML

```
signature BasicIO = sig
        type instream;
        type outstream;
        exception Io of {cause:exn, function:string, name:string}
        val close_in : instream -> unit
        val close_out : outstream -> unit
        val end_of_stream : instream -> bool
        val input : instream * int -> string
        val lookahead : instream -> string
        val open_in : string -> instream
        val open_out : string -> outstream
        val output : outstream * string -> unit
        val std_in : instream
        val std_out : outstream
end;
signature ExtendedIO = sig
        type instream;
        type outstream;
        val can_input : instream * int -> bool
        val flush_out : outstream -> unit
        val open_append : string -> outstream
        val is_term_in : instream -> bool
        val input_line : instream -> string
        val is_term_out : outstream -> bool
        val system : string -> bool
        val get_env : string -> string
        val std_err : outstream
end;
```

Description These are the signatures of the structures that implement SML'90-style I/O. *BasicIO* is open. *ExtendedIO* is not.

ExtendedIO differs from the the original SML'90 in several respects:

- It provides *system* instead of *execute* (which cannot be implemented cleanly on UNIX implementation sof the SML'97 standard basis library, since the SML'90 signature does not give an interface for the caller to reap the executed process).
- It provides *std_err*, which was not in the SML'90 library at all (and is the same as *TextIO.stdErr* in the SML'97 standard basis library).
- It provides *get_env* which is the UNIX *get_env* with non-existent environnent variables returning an empty string.

SML
```
signature PPArray = sig
        exception Subscript
        type 'a array
        val arrayoflist: '_a list -> '_a array
        val array : int * '_a -> '_a array
        val length  : 'a array -> int
        val sub : 'a array * int -> 'a
        val tabulate : int * (int -> '_a) -> '_a array
        val update : 'a array * int * 'a -> unit
end;
```

Description This is the signature of a structure that provides an array datatype compatible with the ProofPower code (independent of the underlying compiler).

SML
```
signature PPString = sig
        val implode : string list -> string;
        val explode : string -> string list;
        exception Ord;
        val ord : string -> int;
        val chr : int -> string;
        val string_of_exn : exn -> string;
end;
```

Description This is the signature of an open structure that provides string functions compatible with the ProofPower code (independent of the underlying compiler).

SML
```
signature PPVector = sig
        exception Subscript
        exception Size
        type '_a vector
        val vector: '_a list -> '_a vector
        val length  : 'a vector -> int
        val sub : 'a vector * int -> 'a
        val tabulate : int * (int -> '_a) -> '_a vector
end;
```

Description This is the signature of a structure that provides a vector (read-only array) datatype compatible with the ProofPower code (independent of the underlying compiler).

SML

```
(*
structure SML97BasisLibrary = struct
        val explode : string -> char list; ...
        structure Array = Array; ...
end;
*)
```

Description This is a structure containing the required structures of the Standard ML '97 Basis Library together with some functions from the '97 standard for the language that are redefined by ProofPower.

It is provided so that these structures can still be accessed when ProofPower defines a structure of the same name as a basis library structure (e.g., "*Char*").

The structure *SML97BasicLibrary.Prelude* contains the functions from the '97 standard for the language that are redefined by ProofPower. If you open this structure and later wish to revert to the ProofPower versions of *explode*, *hd*, etc., open the structures *PPString* and *ListUtilities*.

SYSTEM FACILITIES

3.1 System Control

SML
```
signature SystemControl = sig
```

Description This is the signature of the structure *SystemControl*.

SML
```
val get_flags : unit -> (string * bool) list
val get_int_controls : unit -> (string * int) list
val get_string_controls : unit -> (string * string) list
val get_controls : unit ->
        ((string * bool) list * (string * int) list * (string * string) list)
```

Description These functions return the names and current values of the system flags or controls.

SML
```
val get_flag : string -> bool
val get_int_control : string -> int
val get_string_control : string -> string
```

Description These functions are used to get the values of named control variables of the corresponding types. The parameter gives the name of the control variable.

Errors
```
2011   The name ?0 is not in use as a control variable name
```

Uses This function is for use when adding new facilities to the HOL system which require global control variables.

SML
```
val new_flag :
        {name:string, control:bool ref, default:unit->bool, check:bool -> bool} -> unit
val new_int_control :
        {name:string, control:int ref, default:unit->int, check:int -> bool} -> unit
val new_string_control :
        {name:string, control:string ref, default:unit->string, check:string -> bool} -> unit
```

Description These functions are used to introduce new named control variables of the corresponding types. The *name* parameter gives the name of the new control variable. The *control* component of the parameter gives the variable itself. The *default* component of the parameter is a function which is used by *reset_flag*, *reset_int_control* or *reset_string_control* to reset the value.

After the introduction, users may update the control using one of *set_flag*, *set_int_control* or *set_string_control*.

The *check* component of the parameter is a function to check the validity of the control values, and, if desired, to notify other code of the change in the value. When one of the control setting functions, is called, an error is reported if the *check* function for the control returns *false* when applied to the new value supplied by the caller.

The following message is raised as a warning if the control variable name is already in use. If the user elects to continue, the old control variable is renamed (by decorating it with one or more prime characters) and a new control variable is introduced with the specified name.

Errors
```
2010   The name ?0 is already in use as a control variable name
```

Uses This function is for use when adding new facilities to the HOL system which require global control variables.

SML
```
val pending_reset_control_state : unit -> unit -> unit
```

Description This function is intended for use in system initialisation and shutdown. The binding *val prcs = pending_reset_control_state()*, defines *prcs* as a function which will set the internal state of the *SystemControl* module to the value it had at the time the binding for *prcs* was made. This is used to remember the set-up for controls introduced in a child database. Note that, to avoid problems with stateful user-defined check functions, this function does not attempt to set the values of the controls. The values are, after all, not part of the *SystemControl* module's internal state.

SML
```
val reset_flags : unit -> unit
val reset_int_controls : unit -> unit
val reset_string_controls : unit -> unit
val reset_controls : unit -> unit
```

Description These functions reset the current values of all the system flags or controls in the system, as by *reset_flag*, etc.

SML
```
val reset_flag : string -> bool
val reset_int_control : string -> int
val reset_string_control : string -> string
```

Description These functions are used to reset the values of named control variables of the corresponding types. The parameter gives the name of the control variable. They return the previous value of the control variable.

Errors
```
2011 The name ?0 is not in use as a control variable name
```

Uses This function is for use when adding new facilities to the HOL system which require global control variables.

SML
```
val set_flags : (string * bool) list -> unit
val set_int_controls : (string * int) list -> unit
val set_string_controls : (string * string) list -> unit
val set_controls : ((string * bool) list * (string * int) list * (string * string) list)
        -> unit
```

Description These functions set the current values of the system flags or controls named in the lists. Items that are not mentioned keep their previous values.

SML
```
val set_flag : (string * bool) -> bool
val set_int_control : (string * int) -> int
val set_string_control : (string * string) -> string
```

Description These functions are used to change the values of named control variables of the corresponding types. The first parameter gives the name of the control variable. The second parameter gives the desired new value. They return the previous value of the control variable.

Errors
```
2011 The name ?0 is not in use as a control variable name
2012 Value out of range for control variable ?0
```

Uses This function is the standard means of changing global control variables.

3.2 System Initialisation

SML
```
signature HOLSystem = sig
```

Description This is the signature of the structure *HOLSystem* which contains functions used to end a HOL session and to save the results of a HOL session, as well as two access routes to the UNIX environment to the Standard ML session.

SML
```
signature Initialisation = sig
```

Description This is the signature of the structure *HOLInitialisation* which contains functions which may be used to add and test new start of session functions. These functions are for use by those extending the system.

SML
```
        (* flag: gc_messages; default false *)
```

Description The flag *gc_messages* can be used to turn the Standard ML compiler garbage collector messages on and off (*true* meaning on) providing that facility is supported by the compiler being used. By default, garbage collection messages are turned off.

SML
```
type ICL'DATABASE_INFO_TYPE;
val pp'database_info : ICL'DATABASE_INFO_TYPE;
```

Description Private ProofPower database information, that neither contains information useful to the user, nor should be overwritten by the user. Note that it is not an assignable variable. It is set by *pp'set_database_info*.

SML
```
val get_init_funs : unit -> (unit -> unit) list;
val get_save_funs : unit -> (unit -> unit) list;
```

Description These functions returns the list of functions that have been registered with *new_init_fun* and *new_save_fun*. They are made visible because they are needed to save the state in a child database.

SML
```
val     get_shell_var : string -> string;
```

Description *get_shell_var shvar* will extract the value (as a string), if any, bound to shell environment variable *shvar*. If the variable is not set the empty string will be returned.

SML
```
val init : unit -> unit;
```

Description *init* causes the initialisation functions in the table maintained by *new_init_fun* to be executed, as they would be at the start of a session. The failure of any individual initialisation function will not affect the attempted execution of the others.

Uses Mainly for use in testing extensions to the system.

See Also *new_init_fun*.

Errors
```
36014 Exception caught by init: ?0 (?1)
```

SML
```
val load_files : string list -> bool
```

Description *load_files* takes a list of files and compiles each file (using *use_file*). A message indicating the success or failure is output as each file is processed and a summary is output when all files have been processed. If all the files loaded without any error, *load_files* returns *true* else it returns *false*.

SML
```
val new_init_fun : (unit -> unit) -> unit;
```

Description *new_init_fun* adds a new entry to a table of functions which are invoked at the start of each session. At the beginning of each session, these functions are executed in turn, with the function stored by the most recent use of *new_init_fun* executed last.

SML
```
val new_save_fun : (unit -> unit) -> unit;
```

Description *new_save_fun* adds a new entry to a table of functions which are invoked when the state of a session is saved with *save*, *save_and_quit* or *save_and_exit*. The functions are executed in turn, with the function stored by the most recent use of *new_save_fun* executed last.

SML
```
val pp'reset_database_info: bool -> ICL'DATABASE_INFO_TYPE -> unit;
```

Description This function resets the current system state to a given stored value (which will generally be given by the variable *pp'database_info*), optionally setting the current theory. It is not intended to be called other than in the system start-up code.

SML
```
val pp'set_banner : string OPT -> string;
(* string control: system_banner; default - see below *)
(* string control: user_banner; default - "" *)
```

Description *pp'set_banner* (*Value banner*) will change the core part of the system banner to *banner*, returning the old value. *pp'set_banner Nil* just returns the current value. (The value is held in the string control *system_banner* and can also be changed using *set_string_control* or read *get_string_control*).

The messages below gives the banner, which has elements which may be changed by setting the string controls *system_banner* and *user_banner*. Message 36050 is printed first with *system_banner* as the insertion (?0) followed by message 36051 with insertions giving the latest copyright year (?0) and the *user_banner* (?1). If it is not empty, *user_banner* should begin with a newline character.

Message 36000 gives the value for *system_banner* set in the HOL database, the insertion being the version string taken from the variable *system_version* defined by the make file.

Errors
```
36000 ProofPower ?0 [HOL Database]
36050 === ?0
36051 === Copyright (C) Lemma 1 Ltd. 2000-?0?1
```

SML
```
val pp'set_database_info: unit -> unit;
```

Description This function sets the value of *pp'database_info* so that it describes the current system state. The function is used by *save_and_quit*, and elsewhere, but should not be directly invoked by the user.

SML
```
val pp'theory_hierarchy : pp'Kernel.pp'HIERARCHY_OPT;
```

Description Private ProofPower database information, that neither contains information useful to the user, nor should be overwritten by the user. Note that it is not an assignable variable.

SML
```
val print_banner : unit -> unit;
```

Description Output the system startup banner.

SML
```
val print_status : unit -> unit;
```

Description This command will list:

1. Current theory name;
2. Current proof context name(s);
3. Number of distinct goals to be achieved;
4. Current subgoal label;

SML
```
val quit : unit -> unit
val exit : int -> unit
```

Description *quit*() is used to end a session with the HOL system. In interactive use, the user is warned that the database will not be saved, and asked whether they still wish to quit. The session will be quit if the response is "y", and otherwise the user is returned to the HOL session. If it is used non-interactively, or *use_terminal* (q.v.) is not active, then the session will end without the database being saved.

exit ends the current session of the HOL system with an exit status that is the argument to *exit*. The exit status is available to the calling environment (e.g., as documented in the UNIX manual page for sh(1)). This facility enables the user to flag errors to the outside environment from within ProofPower.

See Also *save_and_quit*, *save_and_exit* to save the database.

SML
```
val save : unit -> unit;
val save_as : string -> unit;
val save_and_quit : unit -> unit;
val save_and_exit : int -> unit;
```

Description *save*() saves the user's current work to disk using the current database name (which is initially derived from the name supplied on the command line when *ProofPower* is invoked using the supplied shell scripts). *save_as name* saves the user's current work to disk under a new name (which becomes the current name used in subsequent calls of *save*()).

save_and_quit() saves the user's current work to disk and then ends the current ProofPower session.

save_and_exit saves the user's current work and then ends the current ProofPower session with an exit status that is the argument to *save_and_exit*. The exit status is available to the calling environment (e.g., as documented in the UNIX manual page for sh(1)). This facility enables the user to flag errors to the outside environment from within ProofPower.

If these function are called from another function rather than at the top-level then the function should be the last side-effecting function call before returning to the top-level, otherwise the behaviour when a new session is started on the saved state will be compiler-dependent.

The state of subsystems such as the subgoal package is preserved between sessions by system-dependent means. The compactification cache is cleared at the end of each session in order to reduce the size of the saved database.

See Also *quit*, *exit*, *clear_compactification_cache*

Errors
```
36010 The database name has not been set
36017 STATE WAS FOUND TO BE INCONSISTENT: state should not be saved
```

Errors If the database cannot be saved then depending on the Standard ML compiler, the function may exit anyway, with a compiler-specific raised error message. The only warning of this is that the start of session text informs the user of the database is read-only at that point in time. This does not happen with Standard ML of New Jersey, which reports the error and then continues the session.

3.3 Warnings

SML
```
signature Warning = sig
```

Description This is the signature of the structure containing the function *warn* which is used to report recoverable error conditions. It also contains the function *comment* which is used to pass comments from the system to the user.

SML
```
val comment : string -> int -> (unit -> string) list -> unit
```

Description *comment* is used to report messages to the user. The parameters are exactly as for *fail* and *error* (q.v.).

Errors
```
10010 *** COMMENT ?0 raised by ?1:
```

SML
```
val warn : string -> int -> (unit -> string) list -> unit
```

Description *warn* is used to report on recoverable error conditions. The parameters are exactly as for *fail* and *error* (q.v.). The behaviour of *warn* depends on the system control flag *ignore_warnings* and on whether or not the system is running interactively, as shown in the following table:

interactive	*ignore_warnings*	Effect
yes	false	the message is reported; the system asks the user whether to continue; if the answer is 'yes' then control returns to the caller of *warn* otherwise an exception is raised.
yes	true	the message is reported and control returns to the caller of *warn*
no	false	the message is reported and an exception is raised
no	true	the message is reported and control returns to the caller of *warn*

Errors
```
10001 *** WARNING ?0 raised by ?1:
10002 Do you wish to continue (y/n)?
10003 execution of ?0 abandoned
```

3.4 Profiling

SML

signature **Profiling** = *sig*

Description The signature contains definitions that may be used to record statistics, e.g., on the number of times certain functions have been called.

SML

(∗ **profiling** − *boolean flag declared by new_flag* ∗)

Description Turns profiling on (if true) or off (if false). Default is false, but flag is true during build of ProofPower-HOL. This should be maintained via the functions of structure *SystemControl*.

SML

```
val prof : string -> unit;
val counts : string -> int OPT;
val get_stats : unit -> int S_DICT;
val set_stats : int S_DICT -> unit;
val print_stats : int S_DICT -> unit;
val init_stats : unit -> unit;
```

Description These five functions provide a simple database facility, associating each name with a count. A call to *prof name* increments, if the flag "profiling" is true, the count for *name*. A call to *counts name* returns the value of the current count for *name*. A call to *get_stats* provides the counting database as an integer dictionary, in order of first name entry into database being first in the dictionary viewed as a list. The function *print_stats* will provide a one line - one entry display of an integer dictionary, in particular the kind of dictionary provided by *get_stats*. A call to *init_stats* initialises all the counts to 0 (which is also the state in which the database starts). A call to *set_stats* will restore a statistics database to a given set of values (such as those given by *get_stats*). The input list must contain no duplicated names.

It is likely that the output of *get_stats* would be best sorted before being printed by *print_stats*.

Uses The intended use of this database is to profile function calls, with the implementer making one call to *prof* per profiled function.

Errors

1020 Input list is ill−formed

3.5 Timing

SML

signature **Timing** = *sig*

Description The signature contains definitions that can be used to measure execution time of ML code.

SML

```
datatype TIMER_UNITS = Microseconds | Milliseconds | Seconds;
type 'b TIMED = {result : 'b, time : int, units : TIMER_UNITS};
val time_app : TIMER_UNITS -> ('a -> 'b) -> 'a -> 'b TIMED;

val reset_stopwatch : unit -> unit ;
val read_stopwatch : TIMER_UNITS -> int;
```

Description The function *time_app* and the associated data types *TIMER_UNITS* and *'a TIMED* may be used to measure the execution time of a function.

In the call *time_app u f x*, *u* specifies the units in which the timing is to be measured, *f* is the function to be timed and *x* gives the argument to be passed to *f*. The return value gives the result of the application *f x* together with the time taken measured in the specified units and a reminder of what the units were.

reset_stopwatch_time and *read_stopwatch_time* give a way of timing sequences of ML commands. *read_stopwatch_time u* returns the elapsed time measured in the units specified by *u* since the last call of *reset_stopwatch_time*. *read_stopwatch_time* will either return a meaningless value or result in arithmetic overflow if *reset_stopwatch_time* has not been called in the current session.

The following points should be born in mind when using these functions:

- The times are “wall-clock” times. Garbage-collection and other overheads will be included.
- Depending on the underlying Standard ML compiler, arithmetic overflow may occur if the units are chosen inappropriately for the time period being measured.
- The functions will themselves introduce a time overhead, which may vary depending on system load and other system-dependent factors.

Errors

1021 Arithmetic overflow in time conversion

INPUT AND OUTPUT

4.1 The Reader/Writer

SML

signature **HOLReaderWriter** = *sig*

Description This structure holds the HOL specific reader writer code. All the errors that the basic reader writer may raise may also be raised by the HOL reader writer.

SML

signature **ReaderWriter** = *sig*

Description File and terminal reading and writing functions.

Errors

5001 End of file found in comment
5002 End of file found in string
5003 Unknown keyword ‘?0‘ after ‘?1‘
5004 Unknown keyword ‘?0‘
5005 Unknown extended character ‘?0‘ (decimal ?1) after ‘?2‘
5006 Unknown extended character ‘?0‘ (decimal ?1)
5007 Unexpected symbol ‘?0‘ (a symbol of type $Invalid$ has been read)
5008 Bracket mismatch, ‘?0‘ found after an opening ‘?1‘
5010 Unknown language requested by symbol ‘?0‘ with language name ‘?1‘
5011 Unknown language requested
5014 Newline found in string after ‘?0‘
5030 End of file in quotation
5032 End of file found in Standard ML quotation
5036 Unknown language ‘?0‘ requested

Several error messages are provided to report faults in the user’s textual input to the ICL HOL system, they may be produced from all of the routines *use_file*, *use_string* and *use_terminal*. Some error messages might be associated with particular routines in the *ReaderWriterSupport* structure but that is incidental to most users, so they are all gathered here.

SML

signature **ReaderWriterSupport** = *sig*

Description A set of declarations that allows the addition of new embedded languages. The HOL language is an example of a language embedded into a basic system that understands Standard ML with extended characters and percent keywords.

SML

(∗ **prompt1** − *boolean flag declared by new_flag, default*: ":>" ∗)
(∗ **prompt2** − *boolean flag declared by new_flag, default*: ":#" ∗)

Description Prompt strings for *use_terminal*. String *prompt1* is used when the reader writer is expecting the first line of a top-level expression, *prompt2* is used for subsequent lines. The strings used here must comprise characters whose decimal codes are in the range 32 to 126 inclusive, but excluding the characters ‘Q’ (i.e., code 81) and ‘%’ (37).

SML
```
(* RW_diagnostics – integer control declared by new_int_control, default: 0 *)
```

Description For reader writer diagnostic purposes.

SML
```
(* use_extended_chars – boolean flag declared by new_flag, default: true *)
```

Description Controls how the writer changes the text output from the PolyML compiler. When *true* extended characters are written, when *false* the corresponding keywords are written.

SML
```
(* use_file_non_stop_mode – boolean flag declared by new_flag, default: false *)
```

Description Makes *use_file* continue reading text (if the flag is *true*) or stop reading (if *false*) from the file after an error is reported by PolyML, this includes both syntax and execution errors. Default is to stop reading.

SML
```
datatype NAME_CLASS
    =    Simple
    |    Starting    of (READER_ENV -> (string * bool)
                            -> string -> bool -> string list
                            -> string list) * string
    |    Middle      of string
    |    Ending      of string
    |    Ignore
    |    Invalid;
```

Description These detail the characteristics of a symbol. *Simple* is used for symbols that may be part of identifiers. *Starting*, *Middle* and *Ending* relate to the symbols position when embedding text of other languages. The function with *Starting* is the reader routine for the particular embedded language. Details of how this function should be written (and of it arguments) are given in the implementation document corresponding to this design. *Ignore* is used for characters that are completely ignored in the input, the extended characters for indexing come in this category. *Invalid* will cause an error message.

See Also Error 5007

SML

```
datatype SYMBOL
      =     SymKnown              of string * bool
                                  * PrettyNames.PRETTY_NAME
      |     SymUnknownChar        of string
      |     SymUnknownKw          of string * bool
      |     SymDoublePercent
      |     SymWhite              of string list
      |     SymCharacter of string
      |     SymEndOfInput
      ;
```

Description *SymKnown* indicates a symbol declared via *add_new_symbols*, if a keyword was read the string hold the characters without the enclosing percents and the boolean is *true*. Otherwise, when an extended character is read the string holds the character and the boolean is *false*.

SymUnknownChar indicates an extended character not declared via *add_new_symbols*.

SymUnknownKw indicates a keyword not declared via *add_new_symbols* or a badly formed keyword with no closing percent sign. The boolean is *true* for a well-formed keyword.

SymDoublePercent indicates an empty keyword, i.e., two adjacent percent signs.

SymWhite indicates a non-empty sequence of formatting characters (space, tab, newline, and formfeed) which are passed as individual characters in reverse order in the string list.

SymEndOfInput indicates an empty string was seen.

All other cases are passed back as a single character in *SymCharacter*.

SML

```
exception TooManyReadEmpties;
```

Description Associated with the reader functions is the exception *TooManyReadEmpties* which is raised when the parser has read the end of the file and has passed the end of file character at least 100 times to the compiler. Raising this exception signifies something has gone wrong in a reader.

SML

```
structure PrettyNames : sig
```

Description A structure within *ReaderWriterSupport* that gathers all the information relating the extended characters and percent keywords understood by the system, together with the interfaces for interrogating and extending the information.

SML
```
type PRETTY_NAME (* = ( string list * string OPT * NAME_CLASS ) *) ;
```

Description Each symbol is defined in a three-element tuple of this type. Elements of the tuple are as follows. First, a non-empty list of the keywords that may be used for this symbol. These keywords exclude the enclosing percent signs. Second, an optional character for the symbol. Third, a value of datatype *NAME_CLASS* indicating the characteristics of the symbol.

The extended character field, when used, contains a single character. It may be the letter "Q" or any character with decimal code greater than 127.

See Also Function *add_new_symbols*, for details of the validation of values of this type.

Example
```
([    "fn",
      "lambda"],    Value "λ",            Simple),
```

SML
```
type READER_ENV (* = {
        advance           : unit -> string,
        look_at_next      : unit -> string,
        push_back         : string -> unit
} *) ;
val skip_and_look_at_next : READER_ENV -> unit -> string;
```

Description All of the parsing functions in the reader writer support use the functions provided in this record type to read characters from the current input stream. Attempting to read characters by any other method will have unpredictable results. The utility function *skip_and_look_at_next* combines *advance* and *look_at_next* discarding the result of *advance*. Some applications will want to use instances of this data type to count line numbers, so pushing back newlines that have not been read is not advisable.

SML
```
type READER_ENV;
type READER_FUNCTION;
```

Description These types are used for reader functions for embedded languages, they are identical to the types of the same name in signature *ReaderWriterSupport*.

See Also Signature *ReaderWriterSupport*.

SML

```
type READER_FUNCTION (*
        = READER_ENV
        -> (string * bool)      (* Starting symbol *)
        -> string               (* Language name *)
        -> string               (* Opening text *)
        -> string list          (* Left hand context *)
        -> string list *);
```

Description The type of the reader functions for embedded languages. The first string argument gives the symbol that started the quotation. For a keyword enclosing percent signs are omitted and the boolean is true. For an extended character the boolean is false. The second string holds the language name without the leading "`%dntext%`" or trailing "`%cantext%`", the default language and type are expanded to give their full names, namely "`HOL`" or "`HOL:`" for the colon form. The third string is text to be included at the start of the quoted text, in the case of a HOL quotation it is the first characters that are to be read by the HOL recogniser. The string list is the left hand context of the call and must be returned with the text of the quotation added to its head.

SML

```
val abandon_reader_writer : unit -> unit;
```

Description Only meaningfully used after *use_terminal* has been called, when it forces an exit from that routine.

SML

```
val add_error_code : int * string list -> string list;
val add_error_codes : int list * string list -> string list;
```

Description For each error number "*nn*" given as the first argument an entry of the form "␣`ERROR__`*nn*␣" is added to the head of the second argument. (Note that "␣" denotes aspace character.)

SML

```
val add_general_reader : string * string * string * READER_FUNCTION -> unit;
val add_specific_reader : string * string * READER_FUNCTION -> unit;
val add_named_reader : string * string * string * READER_FUNCTION -> unit;
```

Description Adds reader functions to the database of known readers. The first strings give the language name, the last string holds the name of a Standard ML constructor which is to be written before the quotation when it occurs in within languages other than Standard ML. Typical values of the last string are "`Lex.Term`" and "`Lex.Type`".

Errors

5033	*Reader already present for language '?0'*
5034	*Improper reader name '?0'*
5035	*Improper reader name '?0' and '?1'*

SML
```
val add_new_symbols : PRETTY_NAME list -> unit;
```

Description Adds details of new symbols to the data structures characterising all known symbols. There is some validation of the symbols added, the list of names should not be empty, the individual names should not contain two adjacent "Q"s and the character field should have a single character which is either a "Q" or has decimal code greater than 127.

Errors

5100	*Keyword '?0' has adjacent 'Q's*
5101	*Empty keyword list*
5102	*Invalid extended character '?0' with keyword '?1'*
5103	*Keyword '?0' duplicated*
5104	*Character '?0' duplicated*

Errors 5100, 5101 and 5102 are issued as warnings against particular parts of the argument value, they do not prevent the other parts from being added to the data structures.

SML
```
val ask_at_terminal : string -> string;
```

Description Asks a question at the terminal by writing out the given string then reading a single line of text which is returned. Characters are read until a newline or end of file is reached, in the first case the the returned string will end with a newline.

Any characters in the type ahead buffer of the terminal input stream before *ask_at_terminal* is called are read and saved (for later analysis by the normal reading functions) before the prompt is output and the response is read.

Errors

5012	*Function 'use_terminal' is not active*
5013	*Input stream is not a terminal, nothing read*

SML
```
val diag_line : string -> unit;
```

Description *diag_line* outputs a string to the standard output stream followed by a new line, after translating it with *translate_for_output*(q.v.). It is intended for use in printing formatted terms, theorems and the like (for which the pretty printer will have included new lines within the string if necessary).

See Also *diag_string*, *raw_diag_line*.

SML
```
val diag_string : string -> unit;
```

Description *diag_string* outputs a string on the standard output stream, after translating it with *translate_for_output*(q.v.). If the string exceeds the value of *get_line_length* it attempts to split the string into tokens, to fit within the line length. A token is taken to be an initial string of spaces, followed by exclusively non-space characters.

See Also *list_diag_string*, *diag_line*, *raw_diag_string*.

SML
```
val expand_symbol : SYMBOL -> string;
```

Description A value of type *SYMBOL* is expanded into the corresponding character string.

SML

```
val find_name : string -> PRETTY_NAME OPT
val find_char : string -> PRETTY_NAME OPT
```

Description Finds the characteristics of a symbol based on its keyword or character. Both functions return *Nil* if the symbol is not known. They return the tuple given to *add_new_symbols* for known symbols.

SML

```
val general_quotation : READER_ENV
        -> (string * bool)   (* Start of quotation symbol *)
        -> string            (* Opening characters *)
        -> bool              (* Context, true => in Standard ML *)
        -> string list       (* Left hand context *)
        -> string list;
val specific_quotation : READER_ENV
        -> (string * bool)   (* Start of quotation symbol *)
        -> string            (* Opening characters *)
        -> bool              (* Context, true => in Standard ML *)
        -> string list       (* Left hand context *)
        -> string list;
val named_quotation : READER_ENV
        -> (string * bool)   (* Start of quotation symbol *)
        -> string            (* Opening characters *)
        -> bool              (* Context, true => in Standard ML *)
        -> string list       (* Left hand context *)
        -> string list;
```

Description Process the text of a quotation and add it to the left hand context given. The opening quotation symbol has been read and is passed as the first string argument, a keyword is passed without its enclosing percent signs and the boolean is true, for an extended character the boolean is false. For general and named quotations the next characters to be read denote the language of the quotation. The boolean argument indicates whether the left hand context is in Standard ML text or in a quotation of another language.

Errors

```
5004  Unknown keyword ‘?0‘
5006  Unknown extended character ‘?0‘ (decimal ?1)
5010  Unknown language requested by symbol ‘?0‘ with language name ‘?1‘
5011  Unknown language requested
5030  End of file in quotation
5031  End of file in language name of general quotation
```

SML

val **get_box_braces** : (*READER_ENV* −> *string list* −> *string list*)
−> *READER_ENV* −> *string list* −> *string list*;
val **get_curly_braces** : (*READER_ENV* −> *string list* −> *string list*)
−> *READER_ENV* −> *string list* −> *string list*;
val **get_round_braces** : (*READER_ENV* −> *string list* −> *string list*)
−> *READER_ENV* −> *string list* −> *string list*;

Description These functions assemble a section of bracketed text. The opening bracket has been read, the first unread character is the first character within the brackets. Each routine reads text upto and including the matching closing bracket. The first argument is the parsing routine for reading items of text within the brackets. The third argument is the left hand context, which is returned with the bracketed text read by these functions, and the enclosing braces. The three pairs of brackets: "`[ ]`", "`{ }`" and "`( )`" are handled by functions *get_box_braces*, *get_curly_braces* and *get_round_braces* respectively.

Errors

5008 Bracket mismatch, '?0' found after an opening '?1'

SML

val **get_HOL_any** : *READER_ENV* −> *string list* −> *string list*

Description Assemble a section of HOL text starting with the first unread character. Text is read up to and including the first unmatched symbol of value *Ending_*. The second argument gives the left hand context, the new text read is added to that context and returned. All the errors that the basic reader writer may raise may also be raised by the HOL reader writer.

See Also Type *READER_ENV*.

SML

val **get_ML_any** : *READER_ENV* −> *string list* −> *string list*

Description Assemble a section of Standard ML text starting with the first unread character. Text is read up to the first semi colon '`;`', unmatched closing bracket or ending keyword. A semi colon will be read and added to the returned text, a closing bracket or ending keyword is left unread for the calling routine. The syntax error where too many closing bracket are presented must be resolved by the outermost routine that calls function. The second argument gives the left hand context, the new text read is added to that context and returned.

Errors

5003 Unknown keyword '?0' after '?1'
5005 Unknown extended character '?0' (decimal ?1) after '?2'
5007 Unexpected symbol '?0' (a symbol of type $Invalid$ has been read)

See Also Type *READER_ENV*.

SML
```
val get_ML_string : READER_ENV -> string list -> string list * int list;
val get_primed_string : READER_ENV -> string list -> string list * int list;
```

Description Assemble a string literal and add it to the left hand context given in the second argument. On entry the opening string quote has been read, exit when the closing string quote has been read. The goal of this routine is to form an equivalent string that can be read by a Standard ML compiler, and to defer as much validation of the string as possible to that compiler. Minimal validation is performed on escape sequences. Well-formed layout sequences (i.e., the sequence "\f..f\") are removed, characters not recognised as formatting ones are retained and wrapped between "\ " and " \" for later checking by the Standard ML compiler. Extended characters are translated to their three digit decimal form.

Function *get_ML_string* reads a Standard ML string.

Function *get_primed_string* reads a string enclosed with single left-hand primes (‘). These are similar to Standard ML strings but with the meanings of the single (‘) and double (") prime characters interchanged.

An end of file found in the string indicates that there is no more input available, and so an immediate failure (error 5002) is raised. Error code 5014 is included to aid in understanding where errors occur, this error is not actually generated until the first non white-space character after the newline is processed. All other errors detected in strings are reported when found, additionally their numbers passed back in the result.

Errors
```
5002 End of file found in string
5014 Newline found in string after ‘?0‘
```

See Also Type *READER_ENV*.

SML
```
val get_percent_name : READER_ENV
        -> string * PrettyNames.PRETTY_NAME OPT * bool;
```

Description Assemble a percent keyword and look it up in the list of known keywords. On entry the opening percent (%) is the first unread character.

The tuple returned contains: (1) the keyword read, but without the percent characters; (2) the symbols entry as given to *add_new_symbols* or *Nil* for an unknown keyword; (3) a flag set true if the keyword had a closing percent character, false otherwise, error reporting is left to the calling functions. Non-alphanumeric keywords may contain the characters "! & $ # + - / : < = > ? @ \ ~ ' ^ | *"

See Also Type *PRETTY_NAME*. Type *READER_ENV*. Function *is_special_char*.

SML
```
val get_use_extended_chars_flag : unit -> bool;
```

Description This function gives the value of the flag *use_extended_chars*.

SML
```
val HOL_lab_prod_reader : READER_FUNCTION;
```

Description This is the reader function for HOL labeled products. It is provided to allow specialised versions of the HOL language to be read, it is not intended to be called directly called by any user code. All the errors that the basic reader writer may raise may also be raised by the HOL reader writer.

SML

val **HOL_reader** : *string* −> *bool* −> *READER_FUNCTION*;

Description This is the HOL reader function, its first argument is the name of the recogniser for the particular aspect of HOL to be recognised. Its second argument indicates whether this reader is considered to be used only at outermost (i.e., Standard ML's top-level): *true* is used for outermost usage, *false* for HOL text that may be used within other expresions. This function is provided to allow specialised versions of the HOL language to be read, it is not intended to be called directly called by any user code. All the errors that the basic reader writer may raise may also be raised by the HOL reader writer.

SML

val **is_same_symbol** : (*string* ∗ *string*) −> *bool*

Description Compare two symbols return true if they are identical, i.e., the same string. Otherwise, look up both with *find_char* and *find_name* then if they are the same symbol return true, if either is not a known symbol or they are not the same symbol return false.

SML

val **is_special_char** : *string* −> *bool*;

Description Checks whether the string contains a single non-alphanumeric character that is allowed in a keyword. Returns *true* if the argument contains exactly one of the characters listed in the description of function *get_percent_name*, otherwise *false* is returned.

See Also Function *get_percent_name*.

SML

val **is_white** : *string* −> *bool*

Description Returns *true* if the string is a single white-space character, *false* otherwise.

SML

val **list_diag_string** : *string list* −> *unit*;

Description *list_diag_string* outputs a list of strings onto the standard output stream, after translating them with *translate_for_output*(q.v.). The strings in the list are concatenated (with spaces to separate them) and then output with *diag_string* (q.v).

See Also *diag_string*, *diag_line*, *list_raw_diag_string*.

SML

val **local_error** : *string* −> *int* −> (*unit* −> *string*) *list* −> *unit*;
val **local_warn** : *string* −> *int* −> (*unit* −> *string*) *list* −> *unit*;

Description An error or warning message is written to the standard output, then the function returns. The arguments are identical in form to functions *error* and *fail* of `DS/FMU/IED/DTD002`.

See Also Functions *error* and *fail*.

SML

val **look_up_general_reader** : *string* ∗ *string* −> (*READER_FUNCTION* ∗ *string*) *OPT*;
val **look_up_specific_reader** : *string* −> (*READER_FUNCTION* ∗ *string*) *OPT*;
val **look_up_named_reader** : *string* ∗ *string* −> (*READER_FUNCTION* ∗ *string*) *OPT*;

Description Looks up readers in the database of known readers. The argument strings are matched against the first string given in the call of the *add_...._reader*, if the reader is known then the corresponding constructor string and reader function are returned. The value *Nil* is returned for an unknown reader.

SML
```
val read_symbol : READER_ENV -> SYMBOL;
```

Description Reads one or more characters and returns a value of type *SYMBOL*. No errors are reported by this routine. The routine reads as many characters as necessary to form a symbol. End of file is returned as a *SymEndOfInput*.

SML
```
val reset_use_terminal : unit -> unit;
```

Description Restores the state that controls *use_terminal* to its default values. N.b., this bypasses the check that *use_terminal* makes on recursive calls (and so could cause a small memory leak if not used with care).

SML
```
val skip_comment : READER_ENV -> unit;
```

Description Skip over a comment which comprises a sequence of characters within which the comment braces '`(*`' and '`*)`' are properly balanced. This routine is entered when the opening round bracket of the comment has been read, the opening asterisk is the first unread character. Note that Standard ML comments separate lexical items thus the calling routine should not simply discard the comment, it might replace the comment with a space character to ensure the lexical items remain separated.

Errors
```
5001 End of file found in comment
```

See Also Type *READER_ENV*.

SML
```
val SML_recogniser : string * string * 'a * string -> 'a;
```

Description This routine is not intended to be directly called by any user code, it is provided to allow the quotation of Standard ML text. The context of use of this routine is that the "macro processing" of the Standard ML quotation "`%<%%dntext%SML%cantext% 42 %>%`" yields the text "`(ReaderWriterSupport.SML_recogniser ("%<%", "SML", 42 , "%>%"))`" which is read by the Standard ML compiler.

Errors
```
5032 End of file found in Standard ML quotation
5050 Incorrect symbols starting or ending of Standard ML quotation: '?0', '?1', '?2'
```

SML
```
val string_of_int3 : int -> string
```

Description The string representation of small positive integers is needed in various places, particularly within Standard ML strings where some characters are denoted by their decimal code in three digits, preceded by a backslash. Function *string_of_int3* gives a three character with leading zeros representation of small positive numbers. In general the routine *PolyML.makestring* cannot be used, if the value last passed to *PolyML.print_depth* is zero then *PolyML.makestring* converts numbers into three dots. The intended use of this function is in building reader writer extensions for other languages. In such places it is intended that the caller only supply suitable arguments, getting this wrong indicates something wrong in the design of the caller. The text of the message anticipates this usage.

Errors
```
5040 DESIGN ERROR:Number ?0 is too big or is negative
```

SML
```
val to_ML_string : string -> string
```

Description Converts characters which are to form part of a string literal into another string which may be read by a Standard ML compiler and which has the same meaning. This is intended to form the string representation of extended characters for passing them through to a Standard ML compiler. Characters other than space, tab and newline which are outside the range 32 to 126 (decimal) inclusive are converted to their four character equivalent of a backslash followed by a three digit decimal number with leading zeroes.

SML
```
val translate_for_output : string -> string;
```

Description Translates a string according to the macro processing rules used when outputting text. The output produced depends on the setting of the control flag *use_extended_chars*, when false the result will have no extended characters, the keyword forms will be used.

SML
```
val use_file : string -> unit;
val use_file1 : string -> unit;
```

Description Both of these functions compile and execute ProofPower-ML (i.e., Standard ML extended to allow mathematical symbols) from the named file. If the file does not exist then the it will read the file with the given name and suffix "`.ML`", if that file does not exist it will try the suffix "`.sml`".

use_file passes the file name string through *translate_for_output* before using it as an operating system file name which is appropriate for file names given as ProofPower-ML strings. The variant *use_file1* uses the string exactly as given.

See Also Error messages given with signature for *ReaderWriter*. Flag *use_file_non_stop_mode*.

Errors
```
5009 Cannot read file '?0' or '?0.ML' or '?0.sml'
```

SML
```
val use_string : string -> unit;
```

Description Read Standard ML with extended characters allowed, from the given string.

See Also Error messages given with signature for *ReaderWriter*.

SML
```
val use_terminal : unit -> unit;
```

Description Read Standard ML with extended characters allowed, from the terminal. This routine takes over the terminal, it handles all exceptions as the outermost level of the ML system. To return to the default PolyML terminal reader use *abandon_reader_writer*.

This routine prompts to the conventions of PolyML but uses the strings "`:> `" and "`:# `", the PolyML prompts do not have the colon. These strings are held as the string controls '*prompt1*' and '*prompt2*' and thus may be altered.

Typing two control-D characters to the terminal prompt, or reading the end-of-file, causes the function *PolyML.quit* to be called.

See Also Error messages given with signature for *ReaderWriter*. Control strings'*prompt1*' and '*prompt2*'.

4.2 Output

SML
signature **SimpleOutput** = *sig*

Description Holds a variety of utility Standard ML functions concerned with simple output. Related facilities may be found in structure ReaderWriter. Function *ask_at_terminal* (q.v) provides for prompted input of text from the terminal.

Strings containing extended characters and strings derived from HOL types and terms should be passed through the ReaderWriter function *translate_for_output* (q.v) before being output. This allows the proper output of keywords and extended characters on both graphic and simple ASCII terminals.

SML
(* **line_length** − *integer control declared by new_int_control* *)

Description An integer control dictating the output's length of line available for printing.

See Also *set_line_length*, *get_line_length*

SML
val **format_list** : ('*a* −> *string*) −> '*a list* −> *string* −> *string*;

Description *format_list formatter items seperator* is used to format a list of items for printing as a string, perhaps for printing. Given *formatter*, a function to format a single item, *items*, a list of items, and *seperator*, a string to separate elements of a multi-element list, the resulting string is the contatenation of the formatted items with interposed separators. The formatted head element of the list becomes the left hand end of the result string.

Example
```
format_list string_of_term [⌜1⌝,⌜x⌝,⌜a ∧ b⌝] ", ";
  --->
val it = "⌜1⌝, ⌜x⌝, ⌜a ∧ b⌝" : string
```

SML
val **get_line_length** : *unit* −> *int*

Description Returns current output line length.

See Also *set_line_length*

SML
val **list_raw_diag_string** : *string list* −> *unit*;

Description *list_raw_diag_string* outputs a list of strings onto the standard output stream. The strings in the list are concatenated (with spaces to separate them) and then output with *raw_diag_string* (q.v).

See Also *raw_diag_string*, *raw_diag_line*, *list_diag_string*.

SML
val **raw_diag_line** : *string* −> *unit*;

Description *raw_diag_line* outputs a string to the standard output stream followed by a new line. It is intended for use in printing formatted terms, theorems and the like (for which the pretty printer will have included new lines within the string if necessary).

See Also *raw_diag_string*, *diag_line*.

SML
```
val raw_diag_string : string -> unit;
```

Description *raw_diag_string* outputs a string on the standard output stream. If the string exceeds the value of *get_line_length* it attempts to split the string into tokens, to fit within the line length. A token is taken to be an initial string of spaces, followed by exclusively non-space characters.

See Also *list_raw_diag_string*, *raw_diag_line*, *diag_string*.

SML
```
val set_line_length : int -> int
```

Description Set the output line length, returning the previous line length. Default length is 80, minimum length 20.

See Also *get_line_length*

Errors
```
1015 line length must be at least 20
```

4.3 HOL Lexical Analysis

SML

```
signature Lex = sig
```

Description This is the signature of the structure which contains the lexical analyser for ICL HOL.

Uses For use by those who wish to extend the system to handle languages other than HOL which have a similar lexical structure.

SML

```
datatype ASSOC = LeftAssoc
               |      RightAssoc;
datatype FIXITY  =    Nonfix
               |      Binder
               |      Infix       of ASSOC * int
               |      Prefix      of int
               |      Postfix     of int;
```

Description These data types are used in the symbol table and elsewhere to give the syntactic status of a name. *Nonfix* means no special status. The integer components are the precedences for infix, prefix or postfix status.

SML

```
datatype HOL_TOKEN = HTAqTm    of TERM
                   | HTAqTy    of TYPE
                   | HTName    of string
                   | HTNumLit  of string
                   | HTString  of string
                   | HTChar    of string
                   | HTBinder  of string
                   | HTInOp    of {name:string, is_type_op:bool,
                                   is_term_op:bool, prec : ASSOC * int}
                   | HTPostOp  of {name:string, prec : int}
                   | HTPreOp   of {name:string, prec : int}
                   | HTAnd
                   | HTBlob
                   | HTColon
                   | HTElse
                   | HTIf
                   | HTIn
                   | HTLbrace
                   | HTLbrack
                   | HTLet
                   | HTLsqbrack
                   | HTRbrace
                   | HTRbrack
                   | HTRsqbrack
                   | HTSemi
                   | HTThen
                   | HTVert
                   | HTEos;
```

Description This is the data type of the output from the HOL lexical analyser.

Uses For use by those who wish to extend the system to handle languages other than HOL which have a similar lexical structure.

SML

```
datatype INPUT = Text      of string
               | String    of string
               | Char      of string
               | Type      of TYPE
               | Term      of TERM
               | Separator of string
               | Error     of int;
```

Description This is the data type of the input to the HOL lexical analyser.

Uses For use by those who wish to extend the system to handle languages other than HOL which have a similar lexical structure.

SML

val is_alnum : string −> bool
val is_copula : string −> bool
val is_digit : string −> bool
val is_macro : string −> bool
val is_punctuation : string −> bool
val is_space : string −> bool
val is_symbolic : string −> bool

Description These functions classify character strings according to their first character. They all return false if the argument is an empty string. The characters for which the various functions return true are shown in the following table.

is_alnum	a letter or a number or the prime character '′'
is_copula	an underscore or the subscription, or superscription characters
is_digit	a decimal digit
is_macro	the character '%' which introduces preprocessor macros
is_punctuation	'(', ')', '{', }', '[', ']', ':', ';', ',', '|', '•' or '$'
is_space	a formatting character, i.e., space, tab, newline etc.
is_symbolic	any character which is not does not fall into any of the above classes

SML

val **lex** : (*string list list*) −> (*string* −> *FIXITY*) −>
INPUT list −> *HOL_TOKEN list*

Description This is the HOL lexical analyser.

The first parameter is the list of (exploded) strings which are to be taken as terminator symbols. Terminators are recognised by looking for the first match in the list, so that if one terminator is a leading substring of another the longer one must come first. No punctuation symbol should appear in a terminator. For HOL this parameter is always obtained by calling the symbol table function *get_terminators*, which maintains the list of terminators sorted in order of decreasing length.

The second parameter is used to classify names as binder, infix, prefix, postfix or nonfix.

The third parameter is the input to be lexically analysed.

Uses For use by those who wish to extend the system to handle languages other than HOL which have a similar lexical structure.

Errors

15001 antiquotation not allowed after '$'
15002 '$' not allowed at end of quotation
15003 lexical analyser or reader/writer error detected (?0)
15004 ill−formed keyword symbol
15005 ?0 is not a valid character literal (must contain exactly one character)
15006 error code ?0 reported by reader/writer

The last of these error messages occurs, e.g., when a keyword symbol has been entered incorrectly and is preceded by a more comprehensive error message from the reader/writer.

SML

```
val num_lit_of_string : string -> (INTEGER * (INTEGER * INTEGER) OPT) OPT;
```

Description The argument to this function should be a string representing a numeric literal (either a natural number, N, or a floating point number with optional, optionally signed, exponent part, $X.Y$ or $X.YeZ$. The result value is *Nil* if the string cannot be interpreted as a numeric literal. Otherwise, the result value is N, or $(XY, P, 0)$ or (XY, PZ), where XY stands for the natural number obtained by concatenating the digit sequences X and Y and P is the number of digits in Y.

4.4 Pretty Printing

SML

signature **PrettyPrinter** = *sig*

SML

(∗ *Flag* **pp_top_level_depth** : *integer control*, *default* −*1* ∗)
(∗ *Flag* **pp_format_depth** : *integer control*, *default* −*1* ∗)

Description These control the depth to which HOL types and terms are printed. Control *pp_top_level_depth* applies to values printed as part of Standard ML top-level expressions. Control *pp_format_depth* applies to values printed by the "*format_* ..." routines. When these controls are negative, types and terms are fully printed, otherwise the value indicates how deeply the expression is printed where zero indicates suppressing the whole type or term. Suppressed types and terms, or parts thereof, are shown as three dots.

See Also Functions *format_term*, *format_term1*, *format_thm*, *format_thm1*, *format_type* and *format_type1*.

SML

(∗ *Flag* **pp_print_assumptions** : *boolean control*, *default true* ∗)

Description This controls whether the assumptions of values of type *THM* are printed. The default is to print assumptions. If assumptions are not printed then each is shown as three dots.

See Also Functions *format_thm* and *format_thm1*.

SML

end (∗ *of signature PrettyPrinter* ∗);

SML

val **format_term** : *bool* −> *TERM* −> *string list*;
val **format_term1** : *bool* −> *int* −> *TERM* −> *string list*;

Description Produce a number of lines, one string per line, containing a pretty printing of the given HOL Term. The text is suitable for directly outputting via the *diag_line* and *diag_string* routines *BasicIO.output*. If the boolean argument is set *false* then the strings produced from terms whose language is the same as that of the current theory will not include the term quotation symbols, in all other cases the term quotation symbols will be included. Line width is given by the integer in *format_term1*, or for *format_term* the current line width (as maintained by *set_line_length*, *q.v.*) is used.

See Also Pretty printer controls: *pp_add_brackets*, *pp_show_HOL_types*, *pp_types_on_binders* and *pp_let_as_lambda*.

SML

val **format_thm** : *THM* −> *string list*;
val **format_thm1** : *int* −> *THM* −> *string list*;

Description Produce a number of lines, one string per line, containing a pretty printing of the given HOL theorem. The text is suitable for directly outputting via the *diag_line* and *diag_string* routines The theorem is printed with a comma separated list of terms for the assumptions, a turnstile and finally the term representing the conclusion. Assumptions in the same language as the conclusion are not enclosed with the term quotation symbols. Other assumptions have term quotation symbols. Line width is given by the integer in *format_term1*, or for *format_term* the current line width (as maintained by *set_line_length*, *q.v.*) is used.

See Also Pretty printer controls: *pp_add_brackets*, *pp_show_HOL_types*, *pp_types_on_binders* and *pp_let_as_lambda*.

SML
```
val format_type : bool -> TYPE -> string list;
val format_type1 : bool -> int -> TYPE -> string list;
```

Description Produce a number of lines, one string per line, containing a pretty printing of the given HOL type. The text is suitable for directly outputting via the *diag_line* and *diag_string* routines If the boolean argument is set *true* then type quotation symbols will be included in the returned strings, when *false* they are excluded. Line width is given by the integer in *format_term1*, or for *format_term* the current line width (as maintained by *set_line_length*, *q.v.*) is used.

See Also Pretty printer control: *pp_add_brackets*.

SML
```
val pp_init : unit -> unit;
```

Description Initialise the pretty printing system so that values of types *TERM*, *TYPE* and *THM* will be prettily printed out as "top level" Standard ML values.

SML
```
val show_type : bool -> int OPT -> OppenFormatting.OPPEN_FUNS
        -> TYPE -> unit;
val show_term : bool -> int OPT -> OppenFormatting.OPPEN_FUNS
        -> TERM -> unit;
val show_thm : int OPT -> OppenFormatting.OPPEN_FUNS
        -> THM -> unit;
```

Description These functions enable programming of Oppen-style pretty-printing for data types that contain embedded types, terms and theorems.

4.5 Theory Lister

SML
```
signature Lister = sig
```

Description This is the signature of the structure *Lister* which contains functions for listing theories.

SML
```
signature ListerSupport = sig
datatype LISTER_SECTION =
        LSBanner      |   LSParents          |   LSChildren
|       LSConsts      |   LSAliases          |   LSUndeclaredAliases
|       LSTypes       |   LSTypeAbbrevs      |   LSUndeclaredTypeAbbrevs
|       LSFixity      |   LSTerminators      |   LSUndeclaredTerminators
|       LSAxioms      |   LSDefns            |   LSThms
|       LSTrailer
|       LSADString of string -> (string list * string)
|       LSADStrings of string -> (string list * string list)
|       LSADThms of string -> (string list * THM) list
|       LSADTerms of string -> (string list * TERM) list
|       LSADTypes of string -> (string list * TYPE) list
|       LSADTables of string -> (string list * string list) list
|       LSADSection of string -> string
|       LSADNestedStructure of string -> (string * LISTER_SECTION list);
```

Description *ListerSupport* is the signature of a structure containing a functions, *gen_theory_lister* and *gen_theory_lister1* for creating variant theory listers, e.g. for languages other than ProofPower-HOL. The data type *ListerSupport.LISTER_SECTION* controls what is listed. Each constructor of this type determines an element of the listing. The first block of constructors for the type *LISTER_SECTION* cause sections of the listing like those produced by the HOL theory lister to be included (except that *LSBanner* uses the first argument to *print*, *output*, or *output1* to compute the contents of the banner heading.) The second block of constructors are for creating application-defined sections of the listing and in each case the constructor takes as its operand a function which is passed the name of the theory being listed as argument. *LSADSection* produces a section header containing the result of applying the argument function to the theory name unless that result is an empty string, in which case it has no effect. The others are for printing (labelled) individual strings (*LSADString*) or columns of strings (*LSADStrings*), or (labelled) lists of theorems, terms, types or rows of strings (*LSADTables*). In each case the first component of (each element of) the result is used as a list of labels for the elements and is printed in the left margin and the second component is indented.

SML
```
(* sorted_listings        - flag - default false *)
(* listing_indent         - integer - control default 2 *)
```

Description These two system control variables influence the behaviour of the functions *list_theory* and *output_theory* which are used to generate theory listings. If *sorted_listings* is false (the default) then items are unsorted, otherwise they are sorted according to *string_order* (q.v.). *listing_indent* sets the indent level of the listings in terms of a number of tabstops, and its default is 2.

Errors
```
33052 integer control '?0' must be greater than zero
```

See Also *output_theory*

SML

```
val gen_theory_lister : LISTER_SECTION list ->
	{	print: (string -> string) -> string -> unit,
		out: (string -> string) -> {theory: string, out_file: string} -> unit,
		out1: (string -> string) -> {theory: string, out_file: string} -> unit};
val gen_theory_lister1 : LISTER_SECTION list ->
	{	print: (string -> string) -> string -> unit,
		out: (string -> string) -> {theory: string, out_file: string} -> unit,
		out1: (string -> string) -> {theory: string, out_file: string} -> unit};
end (* of structure ListerSupport *) (* of structure ListerSupport *);
```

Description The functions *ListerSupport.gen_theory_lister* and *ListerSupport.gen_theory_lister1* are used to create customised theory listers and can also be used to create formatted listings of other kinds.

They return a triple of functions each of which has as its first argument a function to compute the contents of the banner line in the listing from the name of the theory name. Given such an argument, the three components, *print*, *out*, and *out1* deliver results which behave very much like *print_theory*, *output_theory* and *output_theory1*, respectively, as regards where they send the listing and whether or not they insert LaTeX formatting controls in it, but what they put in the listing is determined by the argument to *gen_theory_lister*. This argument is a list of elements of type *LISTER_SECTION*, q.v.

The integer control *listing_indent*and the flag *sorted_listings* control the print of labelled lists of theorems, terms etc. *listing_indent* gives the number of spaces of indent from the left margin of the lists. If *sorted_listings* is true, the lists will be sorted using the concatenation of the labels as the sort key otherwise they are printed in the order supplied.

gen_theory_lister1 is just like *gen_theory_lister* except that it does not check whether the theory exists or whether it is in scope.

SML

```
val output_theory1 : {theory:string, out_file:string} -> unit
```

Description *output_theory1*{*theory* = *thy*, *out_file* = *file*} causes a listing of the theory *thy* to be output to the file *file*. The listing is in a format suited for display on the screen or for viewing with a text editor. The theory must be in scope, i.e. it must be the current theory or one of its ancestors.

See Also *output_theory print_theory*

Errors

```
33050 The theory ?0 is not in scope
33051 There is no theory called ?0
33101 i/o failure on file ?0 (?1)
33102 the theory ?0 does not exist
```

SML
```
val output_theory : {theory:string, out_file:string} -> unit
```

Description *output_theory{theory = thy, out_file = file}* causes a listing of the theory *thy* to be output to the file *file*. The listing is in a format suited for printing using the ICL HOL document preparation system. The theory must be in scope, i.e. it must be the current theory or one of its ancestors.

See Also *output_theory1 print_theory*

Errors
```
33050 The theory ?0 is not in scope
33051 There is no theory called ?0
33101 i/o failure on file ?0 (?1)
33102 the theory ?0 does not exist
```

SML
```
val print_theory : string -> unit
```

Description *print_theory thy* causes a listing of the theory *thy* to be written to the standard output. The listing is in a format suited for display on the screen. The theory must be in scope, i.e. it must be the current theory or one of its ancestors.

Errors
```
33050 The theory ?0 is not in scope
33051 There is no theory called ?0
```

See Also *output_theory output_theory1*

HOL TYPES AND TERMS

5.1 Syntactic Manipulations

It should be noted that the functions documented in this section are drawn from two signatures, *TypesAndTerms* and *pp'TypesAndTerms*.

Since the former includes the latter, an object available under the name *xxx*, say, or in full *TypesAndTerms.xxx*, is also available under the name *pp'TypesAndTerms.xxx*. It is the intention that users should not access objects by names of the form *pp'TypesAndTerms.xxx*. In practice, since the structure *TypesAndTerms* is open, the unqualified name *xxx* will do unless you have redefined the name *xxx*.

SML

```
signature pp'TypesAndTerms = sig
```

Description This provides the type of HOL types: *TYPE*, of HOL terms: *TERM*, and some functions upon them. A user should access all the elements of this signature through signature *DerivedTerms* (q.v).

SML

```
signature TypesAndTerms = sig
```

Description This provides various functions on derived *TERM*s, which are not considered necessary to create the abstract data type *THM*. It also contains, by inclusion, the types, and functions on the types *TERM* and *TYPE* from structure *pp'TypesAndTerms*(q.v.).

SML

```
datatype DEST_SIMPLE_TYPE =
        Vartype of string
|       Ctype of (string * TYPE list);
```

Description This is the type of simple destroyed types, related to the type *TYPE* by *dest_simple_type*(q.v) and *mk_simple_type*(q.v.). The value constructors correspond to type variables and compound types.

SML

```
datatype DEST_SIMPLE_TERM =
        Var of string * TYPE
|       Const of string * TYPE
|       App of TERM * TERM
|       Simpleλ of TERM * TERM;
```

Description This is the simple type of destroyed terms, related to the type *TERM* by *dest_simple_term*(q.v) and *mk_simple_term*(q.v.). The four value constructors represented destroyed variables, constants, applications and simple λ-abstractions respectively.

Uses In writing pattern-matching functions upon HOL terms.

See Also *DEST_TERM*.

SML

datatype **DEST_TERM** = **DVar** *of string* ∗ *TYPE* |
DConst *of string* ∗ *TYPE* |
DApp *of TERM* ∗ *TERM* |
Dλ *of TERM* ∗ *TERM* |
DEq *of TERM* ∗ *TERM* |
D$\Rightarrow$ *of TERM* ∗ *TERM* |
DT |
DF |
D$\neg$ *of TERM* |
DPair *of TERM* ∗ *TERM* |
D$\wedge$ *of TERM* ∗ *TERM* |
D$\vee$ *of TERM* ∗ *TERM* |
D$\Leftrightarrow$ *of TERM* ∗ *TERM* |
DLet *of* ((*TERM* ∗ *TERM*)*list* ∗ *TERM*) |
DEnumSet *of TERM list* |
D$\varnothing$ *of TYPE* |
DSetComp *of TERM* ∗ *TERM* |
DList *of TERM list* |
DEmptyList *of TYPE* |
D$\forall$ *of TERM* ∗ *TERM* |
D$\exists$ *of TERM* ∗ *TERM* |
D$\exists_1$ *of TERM* ∗ *TERM* |
Dϵ *of TERM* ∗ *TERM* |
DIf *of* (*TERM* ∗ *TERM* ∗ *TERM*) |
D$\mathbb{N}$ *of INTEGER* |
DFloat *of INTEGER* ∗ *INTEGER* ∗ *INTEGER* |
DChar *of string* |
DString *of string*;

Description This type is that of a term destroyed using the appropriate derived destructor functions (e.g. *dest_eq*) as well as the primitive ones. The type given to *D*$\varnothing$ and *DEmptyList* is the type of an element of the associated set or list. The type is related to *TERM* by *mk_term* (q.v.) and *dest_term* (q.v)

See Also *DEST_SIMPLE_TERM*

SML

eqtype **TERM**;

Description This is the type of well-formed HOL terms. Objects of this type are manipulated by term constructor, destructor and recogniser functions, such as *mk_app*, *dest_*λ and *is_var*.

SML

eqtype **TYPE**;

Description All HOL terms will be "typed", by associating them with an object of type *TYPE*. A type may either be a type variable or a compound type.

This is not an equality type (i.e. = cannot be used in tests for equality - see =: instead.).

SML

val =$: (*TERM* ∗ *TERM*) −> *bool*;

Description This is the (infix) equality test for HOL terms. It is retained for backwards compatibility — the type of HOL terms is now an equality type.

Instead of equality it is often preferable to test for α-convertibility, using ∼=$

SML
```
val =: : (TYPE * TYPE) -> bool
```

Description This is the (infix) equality test for HOL types. It is retained for backwards compatibility — the type of HOL types is now an equality type.

SML
```
val bin_bool_op : string -> TYPE -> TYPE -> TERM;
```

Description Returns a constant with the given name, and type

⌜:BOOL → BOOL → BOOL⌝

The type arguments are dummies, present only to make the function have an acceptable signature for certain other functions.

SML
```
val BOOL : TYPE;
```

Description The HOL type of truth values:

Definition
```
val BOOL = ⌜:BOOL⌝;
```

See Also Theory "min".

SML
```
val CHAR : TYPE;
```

Description This is the HOL type of single characters.

Definition
```
val CHAR = ⌜:CHAR⌝;
```

See Also Theory "char".

SML
```
val dest_app: TERM -> (TERM * TERM);
```

Description Destroys a function application into the function and argument. Note that many derived term constructs, e.g. all quantifications, are also applications.

Definition
```
dest_app ⌜f t⌝ = (⌜f⌝, ⌜t⌝)
dest_app ⌜∀ x • t⌝ = (⌜$∀⌝, ⌜λ x • t⌝)
```

Errors
```
3010 ?0 is not of form: ⌜t1 t2⌝
```

SML
```
val dest_binder : string -> int -> string -> TERM -> TERM * TERM;
```

Description A generic method of implementing binder destructor functions:

Definition
```
dest_binder area msg binder_nm ⌜binder(λ varstruct• body)⌝ =
        (⌜varstruct⌝, ⌜body⌝)
```

where *binder* is a constant whose name is *binder_nm*. The *varstruct* may be any allowed variable structure.

See Also *dest_simple_binder*

Failure If the term cannot be destroyed, then the error will be from *area*, with a message indexed by *msg*.

SML

val **dest_bin_op** : $string$ −> int −> $string$ −> $TERM$ −> ($TERM$ ∗ $TERM$);

Description *dest_bin_op area msg rator_nm term* first assumes that *term* is of the form ⌜*rator t1 t2*⌝, where *rator* is a constant with name *rator_nm*, and attempts to return the pair (*t1*, *t2*).

Example

dest_bin_op "*dest_*∧" *4032* "∧" ⌜*a* ∧ *b*⌝ = (⌜*a*⌝, ⌜*b*⌝)

If the function fails it will fail with message *msg*, area *area* and with the string form of *term*.

SML

val **dest_char** : $TERM$ −> $string$;

Description Destroy a character literal.

Example

dest_char ⌜'*a*'⌝ = "*a*"

Errors

3024 ?0 is not a character literal

SML

val **dest_const**: $TERM$ −> ($string$ ∗ $TYPE$);

Description This destroys a constant into its name and type.

Errors

3009 ?0 is not a constant

SML

val **dest_ctype** : $TYPE$ −> $string$ ∗ $TYPE$ $list$;

Description Extract the components of a compound type.

Definition

dest_ctype ⌜:(*ty1*,*ty2*,...)*tc*⌝ = ("*tc*", [⌜:*ty1*⌝,⌜:*ty2*⌝,...])
dest_ctype ⌜:*ty tc*⌝ = ("*tc*", [⌜:*ty*⌝])
dest_ctype ⌜:*tc*⌝ = ("*tc*", [])

Errors

3001 ?0 is not a compound type

SML

val **dest_empty_list** : $TERM$ −> $TYPE$;

Description A derived term destructor function for empty lists.

Definition

dest_list ⌜[]:*ty LIST*⌝ = ⌜:*ty*⌝

Errors

4034 ?0 is not of form: ⌜[]⌝

SML

val **dest_enum_set** : $TERM$ −> ($TERM$ $list$);

Description A derived term destructor function for enumerated sets.

Definition

dest_enum_set ⌜{*a*; *b*; ...}⌝ = [⌜*a*⌝, ⌜*b*⌝, ...]

Errors

4011 ?0 is not of form: ⌜{*t1*, ...}⌝

SML
| *val* **dest_eq** : $TERM$ −> ($TERM$ ∗ $TERM$);

Description A derived term destructor function for equations.

Definition
| $dest\_eq$ ⌜$a = b$⌝ = (⌜a⌝,⌜b⌝)
| $dest\_eq$ ⌜$a \Leftrightarrow b$⌝ = (⌜a⌝,⌜b⌝)

Errors
| *3014* *?0 is not of form*: ⌜$t = u$⌝

SML
| *val* **dest_float** : $TERM$ −> $INTEGER$ ∗ $INTEGER$ ∗ $INTEGER$;

Description Destroy a floating point literal.

Definition
| $dest\_float$ ⌜$XXYY.$⌝ = (⌜x⌝, ⌜0⌝, ⌜0⌝)
| $dest\_float$ ⌜$XX.YYeZZ$⌝ = (⌜x⌝, ⌜p⌝, ⌜0⌝)
| $dest\_float$ ⌜$XX.YYeZZ$⌝ = (⌜x⌝, ⌜p⌝, ⌜z⌝)

where x is the natural number with decimal representation $XXYY$, p is the number of digits after the point in $XX.YY$ and z is the integer represented by ZZ (with $p = z = 0$ in the first case and $z = 0$ in the second).

Errors
| *4042* *?0 is not a floating point literal*

SML
| *val* **dest_f** : $TERM$ −> *unit*;

Description This will return () if given the term ⌜F⌝, and otherwise fail.

Errors
| *4037* *?0 is not*: ⌜F⌝

SML
| *val* **dest_if** : $TERM$ −> ($TERM$ ∗ $TERM$ ∗ $TERM$);

Description Destroy a conditional.

Definition
| $dest\_if$ ⌜*if c then y else n*⌝ = (⌜c⌝, ⌜y⌝, ⌜n⌝)

Errors
| *4006* *?0 is not of form*: ⌜*if c then y else n*⌝

SML

val **dest_let** : $TERM$ −> (($TERM$ ∗ $TERM$)$list$ ∗ $TERM$);

Description A derived term destructor function for *let*-terms. See *mk_let* for details of format. The distinction between a local function definition, and a variable structure bound to an abstraction is lost, with both being destroyed to the second form.

Example

```
dest_let(mk_let([(⌜f x⌝, ⌜y⌝)],⌜bdy⌝)) =
        ([(⌜f⌝, ⌜λ x• y⌝)], ⌜bdy⌝)
dest_let(mk_let([(⌜f⌝, ⌜λ x• y⌝)],⌜bdy⌝)) =
        ([(⌜f⌝, ⌜λ x• y⌝)], ⌜bdy⌝)
```

Errors

4009 ?0 is not of form: ⌜*let ... in ...*⌝

dest_let (*mk_let*([], *term*)) will actually fail (unless *term* is already a *let*-term), as apply *mk_let* to ([], *term*)) will just return *term*.

SML

val **dest_list** : $TERM$ −> ($TERM$ $list$);

Description A derived term destructor function for list-terms.

Definition

```
dest_list ⌜[a; b; ...]⌝ = [⌜a⌝, ⌜b⌝, ...]
dest_list ⌜[]⌝ = []
```

Errors

4015 ?0 is not of form: ⌜*[t1,...]*⌝

SML

val **dest_mon_op** : $string$ −> int −> $string$ −> $TERM$ −> $TERM$;

Description *dest_mon_op area msg rator_nm term* assumes that *term* is of the form ⌜*rator t*⌝, where *rator* is a constant with name *rator_nm*, and the function attempts to return *t*.

Example

```
dest_mon_op "dest_¬" 4029 "¬" ⌜¬ t⌝ = ⌜t⌝
```

Failure The failure message for failing to destroy the term will be from area *area*, and will have the text indexed by *msg*, and will have as argument the string form of *term*.

SML

val **dest_multi_¬** : $TERM$ −> (int ∗ $TERM$);

Description *dest_multi_¬ t* will strip ¬ from *t*, returning the number of times, as well as the result. It will return (*0*, *t*) if *t* is either not boolean, or has no negations.

Example

```
dest_multi_¬ ⌜¬(¬ T)⌝ = (2, ⌜T⌝)
```

SML

val **dest_pair** : $TERM$ −> ($TERM$ ∗ $TERM$);

Description A derived term destructor function for pairs.

Definition

```
dest_pair ⌜(t1, t2)⌝ = (⌜t1⌝, ⌜t2⌝)
```

Errors

4003 ?0 is not of form: ⌜*(t1,t2)*⌝

SML
```
val dest_set_comp : TERM -> (TERM * TERM);
```

Description A derived term destructor function for set comprehensions.

Example
```
dest_set_comp ⌜{ x | x > 5}⌝ = (⌜x⌝, ⌜x > 5⌝)
```

Errors
```
4013   ?0 is not of form: ⌜{v | p}⌝
```

SML
```
val dest_simple_binder : string -> int -> string -> TERM -> TERM * TERM;
```

Description Executing *dest_simple_binder area msg binder_nm* ⌜*binder*(λ *var* • *body*)⌝, where *binder* is a constant with the name *binder_nm*, will give (⌜*var*⌝, ⌜*body*⌝).

Example
```
dest_simple_binder "dest_simple_∀" 3032  "∀" ⌜∀ x • t⌝ = (⌜x⌝,⌜t⌝)
```

See Also *dest_binder*

Failure If the term cannot be destroyed, then the error will be from *area*, with a message indexed by *msg*, and argument the string form of *term*.

SML
```
val dest_simple_term : TERM -> DEST_SIMPLE_TERM;
```

Description An injective function, that destroys a term, returning its top-level structure, and the associated constituent parts.

See Also *DEST_SIMPLE_TERM*

SML
```
val dest_simple_type : TYPE -> DEST_SIMPLE_TYPE;
```

Description This function destroys a HOL type into something of type *SIMPLE_DEST_TYPE* (q.v).

SML
```
val dest_simple_∀ : TERM -> (TERM * TERM);
```

Description A derived term destructor function for ∀-terms. It cannot destroy paired abstraction ∀-terms, being the inverse of *mk_simple_∀*.

Definition
```
dest_simple_∀ ⌜∀ var • body⌝ = (⌜var⌝, ⌜body⌝)
```

See Also *dest_∀*

Errors
```
3032   ?0 is not of form: ⌜∀ var • body⌝
```

SML
```
val dest_simple_∃₁ : TERM -> (TERM * TERM);
```

Description A derived term destructor function for simply abstracted ∃_*1*-terms. It may destroy only simple abstraction ∃_*1*-terms, being the inverse of *mk_simple_∃_1*.

Definition
```
dest_simple_∃₁ ⌜∃₁ var• body⌝ = (⌜var⌝, ⌜body⌝)
```

Errors
```
4019   ?0 is not of form: ⌜∃₁ v• t⌝
```

See Also *dest_∃_1*

SML

val **dest_simple_∃** : $TERM$ −> ($TERM$ ∗ $TERM$);

Description A derived term destructor function for ∃-terms. It cannot destroy paired abstraction ∃-terms, being the inverse of *mk_simple_∃*.

Definition

dest_simple_∃ ⌜∃ *var* • *body*⌝ = (⌜*var*⌝, ⌜*body*⌝)

See Also *dest_∃*

Errors

3034 ?0 is not of form: ⌜∃ *var* • *body*⌝

SML

val **dest_simple_λ**: $TERM$ −> ($TERM$ ∗ $TERM$);

Description Destroys a simple λ-abstraction. It cannot destroy paired λ-abstractions, being a inverse of *mk_simple_λ*.

Definition

dest_simple_λ ⌜λ *v* • *t*⌝ = (⌜*v*⌝, ⌜*t*⌝)

See Also *dest_λ*

Errors

3011 ?0 is not of form: ⌜λ *var* • *t*⌝

SML

val **dest_string** : $TERM$ −> *string*;

Description Destroy a string literal.

Example

dest_string ⌜"*abc*"⌝ = "*abc*"

Errors

3025 ?0 is not a string literal

SML

val **dest_term** : $TERM$ −> $DEST\_TERM$

Description This function returns the "best" interpretation of a term in the form of an object of type $DEST\_TERM$. E.g. it will return $DEq($ 1 $2)$ rather than $DComb((\$ = \ 1), 2)$. It will also use the paired abstraction forms of functions in preference to the simple forms, e.g., it uses *dest_λ* not *dest_simple_λ*.

The function assumes that the name of a constant is sufficient to identify it without checking the type, as with, e.g., *dest_bin_op*(q.v).

See Also *mk_term*

SML

val **dest_t** : $TERM$ −> *unit*;

Description This will return () if given the term ⌜T⌝, and otherwise fail.

Errors

4036 ?0 is not: ⌜T⌝

SML

|*val* **dest_vartype** : *TYPE* −> *string*;

Description Extract the name of a type variable:

Definition

|$dest\_vartype$ ⌜: ′tv⌝ = "′tv"

Errors

|*3019* *?0 is not a type variable*
|*3027* *STRING STORE ERROR: cannot translate internal id (?0) to string*

SML

|*val* **dest_var**: *TERM* −> (*string* ∗ *TYPE*);

Description This destroys a term variable into its name and type.

Errors

|*3007* *?0 is not a term variable*

SML

|*val* **dest_∅** : *TERM* −> *TYPE*;

Description A derived term destructor function for empty enumerated sets.

Definition

|$dest\_\varnothing$ ⌜∅:ty SET⌝ = ⌜:ty⌝

Errors

|*4035* *?0 is not of form*: ⌜∅⌝

SML

|*val* **dest_⇔** : *TERM* −> (*TERM* ∗ *TERM*);

Description A derived term destructor function for bi-implications. N.B. this may be successfully applied to boolean equalities.

Definition

|$dest\_\Leftrightarrow$ ⌜$t1 \Leftrightarrow t2$⌝ = (⌜$t1$⌝, ⌜$t2$⌝)

Errors

|*4031* *?0 is not of form*: ⌜$t1 \Leftrightarrow t2$⌝

SML

|*val* **dest_→_type** : *TYPE* −> (*TYPE* ∗ *TYPE*);

Description Extract the two constituent types of a function type.

Definition

|$dest\_\rightarrow\_type$ ⌜:$ty1 \rightarrow ty2$⌝ = (⌜:$ty1$⌝, ⌜:$ty2$⌝)

Errors

|*3022* *?0 is not of form*: ⌜:$ty1 \rightarrow ty2$⌝

SML

|*val* **dest_∧** : *TERM* −> (*TERM* ∗ *TERM*);

Description A derived term destructor function for conjunctions.

Definition

|$dest\_\wedge$ ⌜$t1 \wedge t2$⌝ = (⌜$t1$⌝, ⌜$t2$⌝)

Errors

|*4032* *?0 is not of form*: ⌜$t1 \wedge t2$⌝

SML
```
val dest_∨ : TERM -> (TERM * TERM);
```

Description A derived term destructor function for disjunctions.

Definition
```
dest_∨ ⌜t1 ∨ t2⌝ = (⌜t1⌝, ⌜t2⌝)
```

Errors
```
4027   ?0 is not of form: ⌜t1 ∨ t2⌝
```

SML
```
val dest_¬ : TERM -> TERM;
```

Description A derived term destructor function for negations.

Definition
```
dest_¬ ⌜¬ t⌝ = ⌜t⌝
```

Errors
```
4029   ?0 is not of form: ⌜¬ t⌝
```

SML
```
val dest_⇒ : TERM -> (TERM * TERM);
```

Description A derived term destructor function for implications, returning the antecedent and consequent.

Definition
```
dest_⇒ ⌜a ⇒ b⌝ = (⌜a⌝,⌜b⌝)
```

Errors
```
3016   ?0 is not of form: ⌜t ⇒ u⌝
```

SML
```
val dest_∀ : TERM -> (TERM * TERM);
```

Description A derived term destructor function for ∀-terms. It may destroy a paired abstraction ∀-term, being the inverse of *mk_∀*.

Definition
```
dest_∀  ⌜∀ varstruct• body⌝ = (⌜varstruct⌝, ⌜body⌝)
```

Errors
```
4017   ?0 is not of form: ⌜∀ vs• t⌝
```

SML
```
val dest_∃₁ : TERM -> (TERM * TERM);
```

Description A derived term destructor function for ∃_1-terms. It may destroy paired abstraction ∃_1-terms, being the inverse of *mk_∃_1*.

Definition
```
dest_∃₁ ⌜∃₁ varstruct• body⌝ =
        (⌜varstruct⌝, ⌜body⌝)
```

Errors
```
4021   ?0 is not of form: ⌜∃₁ vs• t⌝
```

See Also *dest_simple_∃_1*

SML

$|$*val* **dest_∃** : *TERM* −> (*TERM* ∗ *TERM*);

Description A derived term destructor function for ∃-terms. It may destroy paired abstraction ∃-terms, being the inverse of *mk_∃*.

Definition

$|$*dest_∃* ⌜∃ *varstruct*• *body*⌝ = (⌜*varstruct*⌝, ⌜*body*⌝)

Errors

$|$*4020* *?0 is not of form*: ⌜∃ *vs*• *t*⌝

See Also *dest_simple_∃*

SML

$|$*val* **dest_×_type** : *TYPE* −> (*TYPE* ∗ *TYPE*)

Description *dest_×_type* ⌜:*ty_1* × *ty_2*⌝ returns (⌜:*ty_1*⌝,⌜:*ty_2*⌝).

Errors

$|$*4018* *?0 is not of the form* ⌜:*ty1* × *ty2*⌝

SML

$|$*val* **dest_ϵ** : *TERM* −> (*TERM* ∗ *TERM*);

Description A derived term destructor function for ϵ-terms.

Definition

$|$*dest_ϵ* ⌜ϵ *varstruct*• *body*⌝ = (⌜*varstruct*⌝, ⌜*body*⌝)

Errors

$|$*4023* *?0 is not of form*: ⌜ϵ *vs*• *t*⌝

SML

$|$*val* **dest_λ**: *TERM* −> (*TERM* ∗ *TERM*);

Description Destroys a λ-abstraction. It can destroy paired λ-abstractions, being an inverse of *mk_λ*.

Definition

$|$*dest_λ* ⌜λ *vs*• *t*⌝ = (⌜*vs*⌝, ⌜*t*⌝)

See Also *dest_simple_λ*

Errors

$|$*4002* *?0 is not of form*: ⌜λ *vs*• *t*⌝

Further details of the errors will be given, before the above exceptions are raised.

SML

$|$*val* **dest_ℕ** : *TERM* −> *INTEGER*;

Description Destroy a numeric literal.

Example

$|$*dest_ℕ* ⌜*5*⌝ = *5*;

Errors

$|$*3026* *?0 is not a numeric literal*

SML

$|$*val* **equality** : *TYPE* −> *TYPE* −> *TERM*;

Description Returns the constant ⌜$ = ⌝ upon terms with the first type argument. The second type is a dummy argument, present only to make the function have an acceptable signature for certain other functions.

SML
```
val frees : TERM -> TERM list;
```

Description Extract the free term variables within the term argument. The resulting variables will be in reverse order of first occurrence (for a term viewed without fixity properties, such as infix variables).

See Also *dest_frees*

SML
```
val gen_vars : TYPE list -> TERM list -> TERM list;
```

Description *gen_vars tyl tml* generates a list of differently named term variables, with the types in *tyl*, whose names are not present within any of the terms in *tml* as variable names.

It will be much faster to make one call to this function with a list of types, than to make the equivalent number of individual calls.

SML
```
val get_variant_suffix : unit -> string;
```

Description Returns the string control *variant_suffix* used to create variant names in *string_variant* (q.v.) and its relatives. The string is set by *set_variant_suffix* (q.v.).

SML
```
val inst_type : ((TYPE * TYPE) list) -> TYPE -> TYPE;
```

Description *inst_type alist type* recursively descends through *type*, replacing any type variables by whatever the association list *alist* associates with them. If the association list does not contain a type variable found in *type*, then that type variable will not be changed. Replaced types are **not** recursively processed by this function.

Errors
```
3019 ?0 is not a type variable
```

SML
```
val inst : TERM list -> (TYPE * TYPE) list -> TERM -> TERM;
```

Description *inst avlist slist term* instantiates the type variables of *term* with the associated types found in *slist*. An element of *slist* will be (*return*, *tv*), where *tv* is a type variable that is to be instantiated to *return*. It will rename bound variables as necessary to prevent name capture problems. It will also not allow free variables to become the same as those in the avoidance list, *avlist*, or to become bound.

It partially evaluates with two arguments.

Errors
```
3007 ?0 is not a term variable
3019 ?0 is not a type variable
3020 Internal error in type instantiation (?0 would become bound)
```

SML
```
val is_app : TERM -> bool;
```

Description Return true only when the term is a function application (i.e. of form ⌜$f\ x$⌝), and false otherwise: no exceptions can be raised. Note that many derived term constructs, e.g. all quantifications, are also applications. Thus *is_app* ⌜$\forall\ x \bullet t$⌝ will return *true*.

SML
|*val* **is_binder** : *string* −> *TERM* −> *bool*;

Description *is_binder binder_nm tm* is true only when *tm* is of the form ⌜*binder*(λ *vs*• *body*)⌝, where *binder* is a constant whose name is *binder_nm*, and *vs* an allowed variable structure, and false otherwise. It cannot raise an exception.

See Also *is_simple_binder*

SML
|*val* **is_bin_op** : *string* −> *TERM* −> *bool*;

Description *is_bin_op rator_nm term* returns true iff. *term* is of the form ⌜rator t1 t2⌝, and *rator* is a constant with name *rator_nm*. It cannot raise an exception.

Example
|*is_bin_op* "∧" ⌜*a* ∧ *b*⌝ = *true*

SML
|*val* **is_char** : *TERM* −> *bool*;

Description Return true only when the term is a character literal (e.g. ⌜'*a*'⌝), and false otherwise: no exceptions can be raised.

SML
|*val* **is_const** : *TERM* −> *bool*;

Description Return true only when the term is a constant, and false otherwise: no exceptions can be raised. Note that even if the constant has not been declared, or has an inappropriate type it will still satisfy this predicate.

SML
|*val* **is_ctype** : *TYPE* −> *bool*;

Description Return true only when the type is a compound type, and false otherwise: no exceptions can be raised. If the argument isn't a compound type then it must be a type variable.

SML
|*val* **is_empty_list** : *TERM* −> *bool*;

Description Return true only when the term is an empty list-term, ⌜[]⌝, and false otherwise: no exceptions can be raised.

SML
|*val* **is_enum_set** : *TERM* −> *bool*;

Description Return true only when the term is an enumerated set (i.e. of form ⌜{*a*; *b*; ...}⌝), and false otherwise: no exceptions can be raised.

SML
|*val* **is_eq** : *TERM* −> *bool*;

Description Return true only when the term is an equation (i.e. of form ⌜*a* = *b*⌝or ⌜*a* ⇔ *b*⌝), and false otherwise: no exceptions can be raised.

SML
|*val* **is_float** : *TERM* −> *bool*;

Description Return true when the term is a floating point literal. and false otherwise: no exceptions are raised.

SML
```
val is_free_in : TERM -> TERM -> bool;
```

Description *is_free_in v term* returns true iff. there is a free occurrence of *v* in *term*. It will raise an exception if the first argument is not a term variable.

Errors
```
3007  ?0 is not a term variable
```

SML
```
val is_free_var_in : (string * TYPE) -> TERM -> bool;
```

Description Given a destroyed term variable, return true only when it is free within the term supplied as a second argument, and false otherwise: no exceptions can be raised.

SML
```
val is_f : TERM -> bool;
```

Description Return true only when the term is ⌜*F* : *BOOL*⌝, and false otherwise: no exceptions can be raised.

SML
```
val is_if : TERM -> bool;
```

Description Return true only when the term is a conditional (i.e. of form ⌜*if a then b else c*⌝), and false otherwise: no exceptions can be raised.

SML
```
val is_let : TERM -> bool;
```

Description Return true only when the term is a *let*-term (i.e. of form ⌜*let x = y in z*⌝), and false otherwise: no exceptions can be raised.

SML
```
val is_list : TERM -> bool;
```

Description Return true only when the term is a list-term (i.e. of form ⌜[*a*; *b*; ...]⌝), and false otherwise: no exceptions can be raised.

SML
```
val is_mon_op : string -> TERM -> bool;
```

Description *is_mon_op rator_nm term* returns true iff. *term* is of the form *rator t*, where *rator* is a constant with name *rator_nm*. It cannot raise an exception.

Example
```
is_mon_op "¬" ⌜¬ t⌝ = ⌜t⌝
```

SML
```
val is_pair : TERM -> bool;
```

Description Return true only when the term is a pair (i.e. of the form ⌜(*a*, *b*)⌝), and false otherwise: no exceptions can be raised.

SML
```
val is_set_comp : TERM -> bool;
```

Description Return true only when the term is a set comprehension (i.e. of form ⌜{*v* | *p*}⌝), and false otherwise: no exceptions can be raised.

SML
|*val* **is_simple_binder** : *string* −> *TERM* −> *bool*;

Description *is_simple_binder binder_nm term* returns true iff. argument *term* is of the form ⌜*binder*(λ *var* • *body*)⌝, where *binder* is a constant with the name *binder_nm*.

See Also *is_binder*

SML
|*val* **is_simple_∀** : *TERM* −> *bool*;

Description A derived term test for simple ∀-terms (i.e. of form ⌜∀ *x* • *t*⌝), not formed with paired abstractions.

See Also *is_∀*

SML
|*val* **is_simple_$\exists_1$** : *TERM* −> *bool*;

Description Return true only when the term is a ∃_1-term (i.e. of form ⌜$\exists_1$ *x* • *t*⌝), formed only by simple abstraction, and false otherwise: no exceptions can be raised.

See Also *is_∃_1*

SML
|*val* **is_simple_∃** : *TERM* −> *bool*;

Description A derived term test for ∃-terms (i.e. of form ⌜∃ *x* • *t*⌝), not formed with paired abstractions.

See Also *is_∃*

SML
|*val* **is_simple_λ** : *TERM* −> *bool*;

Description Is the term a simple λ-abstraction (i.e. of form ⌜λ *x* • *t*⌝).

See Also *is_λ*

SML
|*val* **is_string** : *TERM* −> *bool*;

Description Return true only when the term is a string literal (e.g. ⌜"*abc*"⌝), and false otherwise: no exceptions can be raised.

SML
|*val* **is_type_instance** : *TYPE* −> *TYPE* −> *bool*;

Description *is_type_instance ty_1 ty_2* returns true iff *ty_1* is an instance of *ty_2*. It cannot raise an exception.

SML
|*val* **is_t** : *TERM* −> *bool*;

Description Return true only when the term is ⌜*T* : *BOOL*⌝, and false otherwise: no exceptions can be raised.

SML
|*val* **is_vartype** : *TYPE* −> *bool*;

Description Return true only when the type is a type variable, and false otherwise: no exceptions can be raised. If the argument isn't a type variable then it must be a compound type.

SML
```
val is_var : TERM -> bool;
```

Description Return true only when the term is a variable, and false otherwise: no exceptions can be raised.

SML
```
val is_∅ : TERM -> bool;
```

Description Return true only when the term is an empty enumerated set, ⌜∅⌝, and false otherwise: no exceptions can be raised.

SML
```
val is_⇔ : TERM -> bool;
```

Description Return true only when the term is a bi-implication (i.e. of form ⌜$a \Leftrightarrow b$⌝), and false otherwise: no exceptions can be raised. N.B. this may be successfully applied to boolean equations.

SML
```
val is_→_type : TYPE -> bool;
```

Description Return true only when the type is a function type, i.e. of form ⌜:*ty1* → *ty2*⌝, and false otherwise: no exceptions can be raised.

SML
```
val is_∧ : TERM -> bool;
```

Description Return true only when the term is a conjunction (i.e. of form ⌜$a \land b$⌝), and false otherwise: no exceptions can be raised.

SML
```
val is_∨ : TERM -> bool;
```

Description Return true only when the term is a disjunction (i.e. of form ⌜$a \lor b$⌝), and false otherwise: no exceptions can be raised.

SML
```
val is_¬ : TERM -> bool;
```

Description Return true only when the term is a negation (i.e. of form ⌜$\neg x$⌝), and false otherwise: no exceptions can be raised.

SML
```
val is_⇒ : TERM -> bool;
```

Description Return true only when the term is an implication (i.e. of form ⌜$a \Rightarrow b$⌝), and false otherwise: no exceptions can be raised.

SML
```
val is_∀ : TERM -> bool;
```

Description Return true only when the term is a ∀-term (i.e. of form ⌜∀ *vs* • *t*⌝), possibly formed with paired abstraction, and false otherwise: no exceptions can be raised.

See Also *is_simple_∀*

SML
```
val is_∃₁ : TERM -> bool;
```

Description Return true only when the term is a ∃_*1*-term (i.e. of form ⌜$\exists_1$ *vs* • *t*⌝), possibly formed with paired abstraction, and false otherwise: no exceptions can be raised.

See Also *is_simple_∃_1*

SML
```
val is_∃ : TERM -> bool;
```

Description Return true only when the term is a ∃-term (i.e. of form ⌜∃ *vs* • *t*⌝), possibly formed with paired abstraction, and false otherwise: no exceptions can be raised.

See Also *is_simple_∃*

SML
```
val is_×_type : TYPE -> bool;
```

Description Return true only when the type is a pair type, i.e. of the form: ⌜:*ty_1* × *ty_2*⌝, and false otherwise: no exceptions can be raised.

SML
```
val is_ϵ : TERM -> bool;
```

Description Return true only when the term is a ϵ-term (i.e. of form ⌜ϵ *vs* • *t*⌝), possibly formed with paired abstraction, and false otherwise: no exceptions can be raised.

SML
```
val is_λ : TERM -> bool;
```

Description This function returns true iff. the term is of the form ⌜λ *vs* • *t*⌝. It cannot raise exceptions.

See Also *is_simple_λ*

SML
```
val is_ℕ : TERM -> bool;
```

Description Return true only when the term is a numeric literal (e.g. ⌜*5*⌝), and false otherwise: no exceptions can be raised.

SML
```
val key_mk_const : (E_KEY * TYPE) -> TERM;
val key_dest_const : TERM -> E_KEY * TYPE;
```

Description Internally, the names of constants are represented using efficient dictionary keys. These functions allow the creation and destruction of constants by key rather than by name.

SML
```
val key_mk_ctype : E_KEY * TYPE list -> TYPE;
val key_dest_ctype : TYPE -> E_KEY * TYPE list;
```

Description Internally, the names of type constructors are represented using efficient dictionary keys. These functions allow the creation and destruction of compound types by key rather than by name.

SML
```
val list_mk_app : (TERM * TERM list) -> TERM;
```

Description Applies a function to multiple arguments.

Definition
```
list_mk_app (⌜t⌝, [⌜t1⌝,⌜t2⌝,⌜t3⌝,...]) = ⌜t t1 t2 t3 ...⌝
```

Failure May give rise to the error message from *mk_app*.

SML
```
val list_mk_binder : (TERM * TERM -> TERM) -> (TERM list * TERM)
        -> TERM;
```

Description If *maker* (⌜*vs*⌝,⌜*b*⌝) makes an abstraction ⌜*bind vs* • *b*⌝, then

```
list_mk_binder maker ([⌜vs_1⌝, ⌜vs_2⌝, ...],⌜body⌝)
```

returns ⌜*bind vs_1*• *bind vs_2*• ... • *body*⌝Notice that this can be used for implementing both simple and paired abstractions, with the *vs_i* being variable structures when so allowed, and otherwise variables.

SML
```
val list_mk_bin_op : string -> int -> int ->
        (TYPE -> TYPE -> TERM) -> TERM list -> TERM;
```

Description This function combines a list of terms using the given operator, as if by *mk_bin_op* (q.v). Notice the bracketing in the example.

Example
```
list_mk_bin_op area msg ∧_fun [⌜a⌝, ⌜b ∧ c⌝, ⌜d⌝] =
                ⌜a ∧ ((b ∧ c) ∧ d)⌝
```

where ∧*_fun* takes two (dummy) arguments and returns ⌜$∧⌝.

Errors
```
3017    An empty list argument is not allowed
```

Failure The failure message for failing to combine its arguments will be as *mk_bin_op* for the offending two arguments. If given an empty list the error will be from area *area*, but with message 3017.

SML
```
val list_mk_let : (((TERM * TERM)list)list * TERM) -> TERM
```

Description This generates a nested *let*-term.

Example
```
list_mk_let ([[(⌜x⌝,⌜1⌝)],[(⌜y⌝,⌜2⌝)]],⌜x+y⌝) =
        ⌜let x = 1 in let y = 2 in x+y⌝
```

SML
```
val list_mk_simple_λ : (TERM list * TERM) -> TERM;
```

Description λ-abstract a list of variables from a term.

Definition
```
list_mk_simple_λ ([⌜x1⌝,⌜x2⌝,...],⌜t⌝) = ⌜λ x1 x2 ... • t⌝
```

This function will be implemented using *mk_simple_λ*(q.v), not *mk_λ*.

See Also *list_mk_λ*

Failure May give rise to the error message from *mk_simple_λ*.

SML

|*val* **list_mk_simple_∀** : *TERM list* ∗ *TERM* −> *TERM*;

Description Universally quantify a term with a list of variables.

Definition

|*list_mk_simple_∀* ([⌜*x1*⌝,⌜*x2*⌝,...], ⌜*body*⌝) = ⌜∀ *x1 x2* ... • *body*⌝

This uses *mk_simple_∀* (q.v) to generate its result. Note that giving an empty list paired with a non-boolean will return that term, rather than fail.

See Also *list_mk_∀*

Failure This may give *mk_simple_∀* error messages.

SML

|*val* **list_mk_simple_∃** : *TERM list* ∗ *TERM* −> *TERM*;

Description Existentially quantify a term with a list of variables.

Definition

|*list_mk_simple_∃* ([⌜*x1*⌝,⌜*x2*⌝,...], ⌜*body*⌝) = ⌜∃ *x1 x2* ... • *body*⌝

This uses *mk_simple_∃* (q.v) to generate its result. Note that giving an empty list paired with a non-boolean will return that term, rather than fail.

See Also *list_mk_∃*

Failure This may give *mk_simple_∃* error messages.

SML

|*val* **list_mk_→_type** : *TYPE list* −> *TYPE*;

Description Create the type of a multi-argument function.

Definition

|*list_mk_→_type* [⌜:*ty1*⌝,...,⌜*tyn*⌝] =
| ⌜:*ty1* → ... → *tyn*⌝

The supplied list may not be empty.

Errors

|*3017 An empty list argument is not allowed*

SML

|*val* **list_mk_∧** : *TERM list* −> *TERM*;

Description Conjoin a list of terms:

Definition

|*list_mk_∧* [⌜*a*⌝, ⌜*b*⌝, ⌜*c*⌝,...] = ⌜*a* ∧ *b* ∧ *c* ...⌝

Errors

|*3017 An empty list argument is not allowed*
|*3031 ?0 is not of type* ⌜:*BOOL*⌝

SML

|*val* **list_mk_∨** : *TERM list* −> *TERM*;

Description A function to make a disjunction of a list of terms.

Definition

|*list_mk_∨* [⌜*a*⌝, ⌜*b*⌝, ⌜*c*⌝,...] = ⌜*a* ∨ *b* ∨ *c* ...⌝

Errors

|*3017 An empty list argument is not allowed*
|*3031 ?0 is not of type* ⌜:*BOOL*⌝

SML

val **list_mk_⇒** : $TERM\ list\ {-}{>}\ TERM;$

Description Makes a multiple implication term, using $mk\_{\Rightarrow}$ (q.v.).

Definition

$list\_mk\_{\Rightarrow}\ [\ulcorner t1\urcorner,\ulcorner t2\urcorner,...,\ulcorner tn\urcorner] = \ulcorner t1 \Rightarrow t2 \Rightarrow ... \Rightarrow tn\urcorner$

Note that giving a singleton list containing a non-boolean will return that term, rather than fail.

Errors

3015 ?1 is not of type ⌜:BOOL⌝
3017 An empty list argument is not allowed
3031 ?0 is not of type ⌜:BOOL⌝

SML

val **list_mk_∀** : $TERM\ list * TERM\ {-}{>}\ TERM;$

Description Repeatedly universally quantify a term.

Definition

$list\_mk\_\forall\ ([\ulcorner a\urcorner,\ulcorner b\urcorner,\ulcorner c\urcorner,...], \ulcorner body\urcorner) = \ulcorner\forall\ a\ b\ c\ ...\bullet\ body\urcorner$

This uses $mk\_\forall$ to generate its result.

Failure This may give the errors of $mk\_\forall$.

SML

val **list_mk_∃** : $TERM\ list * TERM\ {-}{>}\ TERM;$

Description Repeatedly existentially quantify a term.

Definition

$list\_mk\_\exists\ ([\ulcorner a\urcorner,\ulcorner b\urcorner,\ulcorner c\urcorner,...], \ulcorner body\urcorner) = \ulcorner\exists\ a\ b\ c\ ...\bullet\ body\urcorner$

This uses $mk\_\exists$ to generate its result.

Failure This may give the errors of $mk\_\exists$.

SML

val **list_mk_ϵ** : $TERM\ list * TERM\ {-}{>}\ TERM;$

Description Repeatedly apply ϵ to a term.

Definition

$list\_mk\_\epsilon\ ([\ulcorner a\urcorner,\ulcorner b\urcorner,\ulcorner c\urcorner,...], \ulcorner body\urcorner) = \ulcorner\epsilon\ a\ b\ c\ ...\bullet\ body\urcorner$

Failure This may give the errors of mk_ϵ.

SML

val **list_mk_λ** : $(TERM\ list * TERM)\ {-}{>}\ TERM;$

Description Repeatedly λ-abstract from a term.

Definition

$list\_mk\_\lambda\ ([\ulcorner a\urcorner,\ulcorner b\urcorner,\ulcorner c\urcorner,...], \ulcorner body\urcorner) = \ulcorner\lambda\ a\ b\ c\ ...\bullet\ body\urcorner$

This function is implemented using mk_λ, not mk_simple_λ.

See Also $list\_mk\_simple\_\lambda$

Failure May give rise to the error message from mk_λ.

SML
```
val list_term_union : (TERM list list) -> TERM list;
```

Description Take the union of a number of lists of terms viewed as sets, removing any α-convertible duplicates.

See Also *list_union* for precise ordering of result.

SML
```
val list_variant : TERM list -> TERM list -> TERM list;
```

Description *list_variant stoplist vlist* returns a list of variants of the list of variables *vlist*, whose names are not present in the *stoplist*, which is also a list of term variables. No names are duplicated, the function returning one new variable for each member of *vlist*. The variants are generated by sufficient appending of the variant string (see *set_variant_string*).

Errors
```
3007   ?0 is not a term variable
```

SML
```
val mk_app : (TERM * TERM) -> TERM;
```

Description This produces a function application.

Definition
```
mk_app (⌜f⌝, ⌜t⌝) = ⌜f t⌝
```

Note that many derived term constructs, e.g. all quantifications, are also applications. Thus

Example
```
mk_app (⌜$∀⌝, ⌜λ x • t⌝) = ⌜∀ x • t⌝
```

Errors
```
3005   Cannot apply ?0 to ?1 as types are incompatible
3006   Type of ?0 not of form ⌜:ty1 → ty2⌝
```

SML
```
val mk_binder : string -> int -> (TYPE -> TYPE -> TERM) ->
        (TERM * TERM) -> TERM;
```

Description A generic method of implementing binder constructor functions:

Definition
```
mk_binder area msg binder_nm (⌜varstruct⌝, ⌜body⌝) =
        ⌜binder′(λ varstruct• body)⌝ =
        ⌜binder′ varstruct• body⌝
```

binder′ is formed by applying *binder* to the types of the *varstruct* and *body*. *varstruct* may be any allowed variable structure.

See Also *mk_simple_binder*

Errors
```
4016   ?0 is not an allowed variable structure
```

Failure If the term cannot be made, then the error will be from *area*, with a message indexed by *msg*. If the first term argument is not an allowed variable structure then failure 4016 is raised from area *area*.

SML

$$val\ \mathbf{mk\_bin\_op} : string\ {-}{>}\ int\ {-}{>}\ int\ {-}{>}\ (TYPE\ {-}{>}\ TYPE\ {-}{>}\ TERM)\ {-}{>}\ (TERM * TERM)\ {-}{>}\ TERM;$$

Description *mk_bin_op area msg1 msg2 rator_fn* (t_1, t_2) attempts to form $\ulcorner t\_1\ rator\ t\_2 \urcorner$. $rator'$ is gained by applying *rator_fn* to the types of *t_1* and *t_2*.

Example

$$mk\_bin\_op\ \texttt{"}mk\_\wedge\texttt{"}\ 3031\ 3015\ (fn\ \_\ {=}{>}\ fn\ \_\ {=}{>}\ \ulcorner\$\wedge\urcorner)\ (\ulcorner a\urcorner, \ulcorner b\urcorner) = \ulcorner a \wedge b\urcorner$$

Failure The failure message for failing to apply *rator* to the first term will be from area *area*, and will have the text indexed by *msg1*, with the two terms as strings for arguments. If the failure is from applying the rators plus first term to the second term the error message will be from area *area*, and will have the text indexed by *msg2*, with the two terms as strings for arguments. It is not unusual for one of these strings of terms to be thrown away by the message *msg2* provided by the caller of this function.

SML

$$val\ \mathbf{mk\_char} : string\ {-}{>}\ TERM;$$

Description Construct a character literal.

Example

$$mk\_char\ \texttt{"}a\texttt{"} = \ulcorner`a`\urcorner$$

Errors

3023	*String ?0 is not a single character*

SML

$$val\ \mathbf{mk\_const} : (string * TYPE)\ {-}{>}\ TERM;$$

Description This produces a constant.

Definition

$$mk\_const(\texttt{"}c\texttt{"}, \ulcorner{:}ty\urcorner) = \ulcorner c : ty\urcorner$$

The function makes no checks against the declaration of the constant, the declaration of the type constructors of the type supplied, or the appropriateness of the type supplied: see *get_const_info* (q.v.). However it will not form constants whose types clash with those constants required by the implementation of the abstract data type *THM* (q.v.). These are $=$, $\Rightarrow$, $\forall$, and $\exists$.

Errors

3002	*Type of constant with name "=" must be of form:*$\ulcorner ty1 \rightarrow ty1 \rightarrow BOOL\urcorner$
3003	*Type of constant with name "⇒" must be of form:*$\ulcorner BOOL \rightarrow BOOL \rightarrow BOOL\urcorner$
3004	*Type of constant with name ?0 must be of form:*$\ulcorner (ty1 \rightarrow BOOL) \rightarrow BOOL\urcorner$

SML

$$val\ \mathbf{mk\_ctype} : string * TYPE\ list\ {-}{>}\ TYPE;$$

Description Create a compound type from a type constructor and sufficient arguments. The function makes no checks against the declaration or arity of the type constructor or the type arguments: see *get_type_info* (q.v.).

Definition

$$mk\_ctype\ (\texttt{"}tc\texttt{"}, [\ulcorner{:}ty1\urcorner, \ulcorner{:}ty2\urcorner, ...]) = \ulcorner{:}(ty1, ty2, ...)tc\urcorner$$
$$mk\_ctype\ (\texttt{"}tc\texttt{"}, [\ulcorner{:}ty\urcorner]) = \ulcorner{:}ty\ tc\urcorner$$
$$mk\_ctype\ (\texttt{"}tc\texttt{"}, []) = \ulcorner{:}tc\urcorner$$

SML

```
val mk_empty_list : TYPE -> TERM
```

Description A derived term constructor function for generating an empty list term with elements of a given type.

Definition

```
mk_empty_list ⌜: ty⌝ = ⌜[] : ty LIST⌝
```

See Also *mk_list*

SML

```
val mk_enum_set : TERM list -> TERM
```

Description A derived term constructor function for generating enumerated sets. The argument is a list of the members of the set. The type of a set of elements of type ⌜:*TY*⌝ is ⌜:*TY SET*⌝. If the term list is empty the function will fail (see *mk_∅*). The set must be of terms with the same HOL type.

Definition

```
mk_enum_set [⌜a⌝, ⌜b⌝, ...] = ⌜{a; b; ...}⌝
```

Errors

```
3012   ?0 and ?1 do not have the same types
3017   An empty list argument is not allowed
```

SML

```
val mk_eq : (TERM * TERM) -> TERM;
```

Description A derived term constructor function for generating equations.

Definition

```
mk_eq (⌜a⌝,⌜b⌝) = ⌜a = b⌝
mk_eq (⌜a:BOOL⌝,⌜b:BOOL) = ⌜a ⇔ b⌝
```

Errors

```
3012   ?0 and ?1 do not have the same types
```

SML

```
val mk_float : INTEGER * INTEGER * INTEGER -> TERM;
```

Description Make a floating point literal.

Definition

```
mk_float (⌜x⌝, 0, ⌜0⌝) = ⌜XX.⌝
mk_float (⌜x⌝, ⌜p⌝, ⌜0⌝) = ⌜XX.YY⌝
mk_float (⌜x⌝, ⌜p⌝, ⌜z⌝) = ⌜XX.YYeZZ⌝
```

where $XX.YY$ is the decimal representation of $x \times 10^{-p}$ and ZZ is the decimal representation of z (with $p = z = 0$ in the first case and $z = 0$ in the second).

Errors

```
4041   the mantissa of a HOL floating point literal must be non-negative
```

SML

```
val mk_f : TERM;
```

Description The term ⌜*F* : *BOOL*⌝.

SML
```
val mk_if : (TERM * TERM * TERM) -> TERM;
```

Description Make a conditional.

Definition
```
mk_if (⌜c⌝, ⌜y⌝, ⌜n⌝) = ⌜if c then y else n⌝
```

Errors
```
3012   ?0 and ?1 do not have the same types
3031   ?0 is not of type ⌜:BOOL⌝
```

SML
```
val mk_let : ((TERM * TERM)list * TERM) -> TERM
```

Description A derived term constructor function for generating *let*-terms. The arguments may have any form allowed by ICL HOL Concrete Syntax. Thus they may be variable structures formed by pairing, or single clause, non-recursive functions, whose arguments may only be variable structures formed by pairing.

Example
```
mk_let ([],⌜x⌝) = ⌜x⌝
mk_let([(⌜x⌝, ⌜1⌝)], ⌜x+1⌝) = ⌜let x = 1 in x + 1⌝
mk_let([(⌜x⌝, ⌜1⌝),(⌜y⌝, ⌜2⌝)], ⌜x+y⌝) =
        ⌜let x = 1 and y = 2 in x + y⌝
mk_let([(⌜(x,y)⌝, ⌜(1,2)⌝)], ⌜x+y⌝) =
        ⌜let (x,y) = (1,2) in x + y⌝
mk_let([(⌜(x,y)⌝, ⌜(1,2)⌝)], ⌜x+y⌝) =
         ⌜let (x,y) = (1,2) in x + y⌝
mk_let([(⌜f (x,y)⌝, ⌜(1,2)⌝)], ⌜x+y⌝) =
        ⌜let f = λ (x,y)• (1,2) in x + y⌝
```

Errors
```
3012   ?0 and ?1 do not have the same types
4007   ?0 is not a well−formed LHS for mk_let
```

SML
```
val mk_list : TERM list -> TERM
```

Description A derived term constructor function for generating list-terms. The argument is a list of the members of the list. If the term list is empty the function will fail (see *mk_empty_list*). The list must be of terms with the same HOL type.

Definition
```
mk_list [⌜a⌝, ⌜b⌝, ...] = ⌜[a; b; ...]⌝
```

Errors
```
3012   ?0 and ?1 do not have the same types
3017   An empty list argument is not allowed
```

SML

val **mk_mon_op** : *string* −> *int* −> (*TYPE* −> *TERM*) −>
TERM −> *TERM*;

Description *mk_mon_op area msg rator_fn* ⌜*rand*⌝ attempts to form the term ⌜*rator rand*⌝. ⌜*rator*⌝ is gained by applying *rator_fn* to the type of ⌜*rand*⌝.

Example

mk_mon_op "*mk_¬*" *3031* (*fn* _ => ⌜$¬⌝) ⌜*t*:*BOOL*⌝ = ⌜¬ *t*⌝

Failure The failure message for failing to apply *rator* to its arguments will be from area *area*, and will have the text indexed by *msg*.

SML

val **mk_multi_¬** : (*int* ∗ *TERM*) −> *TERM*;

Description *mk_multi_¬* (n, t) will apply the constructor *mk_¬* n times to t.

Example

mk_multi_¬ (*2*, ⌜*T*⌝) = ⌜¬(¬ *T*)⌝

Errors

3031 *?0 is not of type* ⌜:*BOOL*⌝
4030 *?0 is negative*

SML

val **mk_pair** : (*TERM* ∗ *TERM*) −> *TERM*;

Description A derived term constructor function for generating pairs.

Definition

mk_pair(⌜*t1*⌝, ⌜*t2*⌝) = ⌜(*t1*, *t2*)⌝

SML

val **mk_set_comp** : (*TERM* ∗ *TERM*) −> *TERM*

Description A derived term constructor function for generating set comprehensions.

Example

mk_set_comp (⌜*x*⌝, ⌜*x* > *5*⌝) = ⌜{ *x* | *x* > *5*}⌝

Errors

3015 *?1 is not of type* ⌜:*BOOL*⌝
4016 *?0 is not an allowed variable structure*

SML

val **mk_simple_binder** : *string* −> *int* −> (*TYPE* −> *TYPE* −> *TERM*) −>
(*TERM* ∗ *TERM*) −> *TERM*;

Description *mk_simple_binder area msg binder_fn* (*var*, *body*) generates the term:

⌜*binder*(λ*var* • *body*)⌝

where *binder* is *binder_fn* applied to the types of *var* and *body*. *var* must be a term variable.

See Also *mk_binder*

Errors

3007 *?0 is not a term variable*

Failure If the term cannot be made, then the error will be from *area*, with a message indexed by *msg*, and the two terms as string arguments. If the first of the pair of terms is not a variable then error 3007 will be given from area *area*.

SML
val **mk_simple_term** : *DEST_SIMPLE_TERM -> TERM*;

Description Create a well-formed TERM from a statement of a top-level structure, and the associated constituent parts.

It makes the same checks as *mk_const*, *mk_app*, etc(q.v.), and gives the same error messages as these if there is a failure.

See Also *DEST_SIMPLE_TERM*

Errors
3005 Cannot apply ?0 to ?1 as types are incompatible
3006 Type of ?0 not of form ⌜:ty1 → ty2⌝
3007 ?0 is not a term variable

SML
val **mk_simple_type** : *DEST_SIMPLE_TYPE -> TYPE*;

Description This function constructs a HOL type from something of type *SIMPLE_DEST-_TYPE* (q.v).

SML
val **mk_simple_∀** : *(TERM * TERM) -> TERM*;

Description A derived term constructor function for generating simple ∀-terms.

Definition
mk_simple_∀ (⌜var⌝, ⌜body⌝) = ⌜∀ var • body⌝

var must be a term variable.

See Also *mk_∀*

Errors
3007 ?0 is not a term variable
3015 ?1 is not of type ⌜:BOOL⌝

SML
val **mk_simple_∃$_1$** : *(TERM * TERM) -> TERM*;

Description A derived term constructor function for generating simply abstracted ∃_*1*-terms.

Definition
mk_simple_∃$_1$ (⌜var⌝, ⌜body⌝) =
⌜∃$_1$ var• body⌝

var must be a variable.

Errors
3007 ?0 is not a term variable
3015 ?1 is not of type ⌜:BOOL⌝

See Also *mk_∃_1*

SML
|*val* **mk_simple_∃** : (*TERM* ∗ *TERM*) −> *TERM*;

Description A derived term constructor function for generating simple ∃-terms.

Definition
|*mk_simple_∃* (⌜*var*⌝, ⌜*body*⌝) = ⌜∃ *var* • *body*⌝

var must be a term variable.

See Also *mk_∃*

Errors
|*3007 ?0 is not a term variable*
|*3015 ?1 is not of type* ⌜:*BOOL*⌝

SML
|*val* **mk_simple_λ** : (*TERM* ∗ *TERM*) −> *TERM*;

Description This produces a simple λ-abstraction. It may only abstract variables.

Definition
|*mk_simple_λ* (⌜*v*⌝, ⌜*t*⌝) = ⌜λ *v* • *t*⌝

See Also *mk_λ*

Errors
|*3007 ?0 is not a term variable*

SML
|*val* **mk_string** : *string* −> *TERM*;

Description Construct a string literal.

Example
|*mk_string* "*abc*" = ⌜"*abc*"⌝

SML
|*val* **mk_term** : *DEST_TERM* −> *TERM*

Description Create a term from a derived term. It is an inverse to *dest_term* (q.v), and therefore understands how to handle paired abstractions.

The function is implemented using the individual primitive and derived term constructors (e.g. *mk_const* and *mk_∀*), with what checks they use.

Failure This function will fail with the same messages as the appropriate term constructor functions.

SML
|*val* **mk_t** : *TERM*;

Description The term ⌜*T* : *BOOL*⌝.

SML
|*val* **mk_vartype** : *string* −> *TYPE*;

Description Create a HOL type variable from a string:

Definition
|*mk_vartype* "′*tv*" = ⌜: ′*tv*⌝

SML

val **mk_var** : (*string* ∗ *TYPE*) −> *TERM*;

Description This produces a term variable.

The function makes no checks against the declaration of the subtypes of the type supplied.

Definition

mk_var("*v*",⌜:*ty*⌝) = ⌜*v* : *ty*⌝

SML

val **mk_∅** : *TYPE* −> *TERM*;

Description A derived term constructor function for generating an empty (enumerated) set with elements of a given type.

Definition

mk_∅ ⌜: *ty*⌝ = ⌜∅ : *ty SET*⌝

See Also *mk_enum_set*

SML

val **mk_⇔** : (*TERM* ∗ *TERM*) −> *TERM*;

Description A derived term constructor function for generating bi-implications.

Definition

mk_⇔ (⌜*t1*⌝, ⌜*t2*⌝) = ⌜*t1* ⇔ *t2*⌝

Errors

3015 ?1 is not of type ⌜:*BOOL*⌝
3031 ?0 is not of type ⌜:*BOOL*⌝

SML

val **mk_→_type** : (*TYPE* ∗ *TYPE*) −> *TYPE*;

Description Create a function type from two types. A function type is just a kind of compound type.

Definition

mk_→_type (⌜:*ty1*⌝, ⌜:*ty2*⌝) =
    mk_ctype("→",[⌜:*ty1*⌝,⌜:*ty2*⌝]) = ⌜:*ty1* → *ty2*⌝

SML

val **mk_∧** : (*TERM* ∗ *TERM*) −> *TERM*;

Description A derived term constructor function for generating conjunctions.

Definition

mk_∧ (⌜*t1*⌝, ⌜*t2*⌝) = ⌜*t1* ∧ *t2*⌝

Errors

3015 ?1 is not of type ⌜:*BOOL*⌝
3031 ?0 is not of type ⌜:*BOOL*⌝

SML

val **mk_∨** : (*TERM* ∗ *TERM*) −> *TERM*;

Description A derived term constructor function for generating disjunctions.

Definition

mk_∨ (⌜*t1*⌝, ⌜*t2*⌝) = ⌜*t1* ∨ *t2*⌝

Errors

3015 ?1 is not of type ⌜:*BOOL*⌝
3031 ?0 is not of type ⌜:*BOOL*⌝

SML
```
val mk_¬ : TERM -> TERM;
```

Description A derived term constructor function for generating negations.

Definition
```
mk_¬ ⌜t⌝ = ⌜¬ t⌝
```

Errors
```
3031 ?0 is not of type ⌜:BOOL⌝
```

SML
```
val mk_⇒ : (TERM * TERM) -> TERM;
```

Description A derived term constructor function for generating implications. It takes two arguments: the antecedent and the consequent.

Definition
```
mk_⇒ (⌜a⌝,⌜b⌝) = ⌜a ⇒ b⌝
```

Errors
```
3015 ?1 is not of type ⌜:BOOL⌝
3031 ?0 is not of type ⌜:BOOL⌝
```

SML
```
val mk_∀ : (TERM * TERM) -> TERM;
```

Description A derived term constructor function for generating ∀-terms.

Definition
```
mk_∀ (⌜varstruct⌝, ⌜body⌝) = ⌜∀ varstruct• body⌝
```

varstruct may be any allowed variable structure.

Errors
```
3015 ?1 is not of type ⌜:BOOL⌝
4016 ?0 is not an allowed variable structure
```

See Also *mk_simple_∀*

SML
```
val mk_∃₁ : (TERM * TERM) -> TERM;
```

Description A derived term constructor function for generating ∃_1-terms.

Definition
```
mk_∃₁ (⌜varstruct⌝, ⌜body⌝) =
        ⌜∃₁ varstruct• body⌝
```

varstruct may be any allowed variable structure.

Errors
```
3015 ?1 is not of type ⌜:BOOL⌝
4016 ?0 is not an allowed variable structure
```

See Also *mk_∃_1*

SML

val **mk_∃** : (*TERM* ∗ *TERM*) −> *TERM*;

Description A derived term constructor function for generating ∃-terms.

Definition

mk_∃ (⌜*varstruct*⌝, ⌜*body*⌝) = ⌜∃ *varstruct*• *body*⌝

varstruct may be any allowed variable structure.

Errors

3015 ?1 is not of type ⌜*:BOOL*⌝
4016 ?0 is not an allowed variable structure

See Also *mk_simple_∃*

SML

val **mk_×_type** : (*TYPE* ∗ *TYPE*) −> *TYPE*

Description *mk_×_type* (⌜*:ty_1*⌝,⌜*:ty_2*⌝) returns a pair type: ⌜*:ty_1* × *ty_2*⌝.

SML

val **mk_ϵ** : (*TERM* ∗ *TERM*) −> *TERM*;

Description A derived term constructor function for generating ϵ-terms.

Definition

mk_ϵ (⌜*varstruct*⌝, ⌜*body*⌝) = ⌜ϵ *varstruct*• *body*⌝

varstruct may be any allowed variable structure.

Errors

3015 ?1 is not of type ⌜*:BOOL*⌝
4016 ?0 is not an allowed variable structure

SML

val **mk_λ** : *TERM* ∗ *TERM* −> *TERM*

Description This creates a λ-abstraction of an allowed variable structure from a term.

Example

mk_λ (⌜*x*⌝ , ⌜*x* + *y*⌝) = ⌜λ *x*• *x* + *y*⌝
mk_λ (⌜(*x*, *y*)⌝, ⌜*x* + *y*⌝) = ⌜λ (*x*, *y*)• *x* + *y*⌝
mk_λ (⌜((*x1*,*x2*), (*y1*,*y2*))⌝, ⌜*x2* + *y2*⌝) = ⌜λ ((*x1*,*x2*), (*y1*,*y2*))• *x2* + *y2*⌝

See Also *mk_simple_λ*

Errors

4016 ?0 is not an allowed variable structure

SML

val **mk_ℕ** : *INTEGER* −> *TERM*;

Description Construct a numeric literal: the argument may not be negative.

Example

mk_ℕ 5 = ⌜*5*⌝

Errors

3021 ?0 should be 0 or positive

SML

val **quantifier** : *string* −> *TYPE* −> *TYPE* −> *TERM*;

Description *quantifier name type dummy* returns a constant, with the given name, and type ⌜:(*type*→*BOOL*)→*BOOL*⌝, This is an appropriate type for binders. The dummy is present only to make the function have an acceptable signature for certain other functions.

SML
val **rename** : (*string* ∗ *TYPE*) −> *string* −> *TERM* −> *TERM*;

Description *rename* (*oname*, *type*) *cname term* returns a term based on *term*, but with any free variables with name *oname*, and type *type* renamed to *cname*.

SML
val **set_variant_suffix** : *string* −> *string*;

Description Sets the string control *variant_suffix* used to create variant names in *string-_variant* (q.v.) and its relatives. The string is initially a single prime character. The function returns the previous setting of the control.

Errors
3028 string may not be empty

SML
val **string_of_term** : *TERM* −> *string*;

Description This returns a display of a term in the form of a string, with no inserted new lines, suitable for use with *diag_string* and *fail*.

See Also *format_term* is a formatted string display of a term.

SML
val **string_of_type** : *TYPE* −> *string*;

Description This returns a display of a type in the form of a string, with no inserted new lines, suitable for use with *diag_string* and *fail*.

See Also *format_type* is a formatted string display of a type.

SML
val **string_variant** : *string list* −> *string* −> *string*;

Description *string_variant vlist name* returns a string that is a different from any name in *vlist*. Variants are formed by repeatedly appending the variant string(see *set_variant_string*) to the *name*. Note that *string_variant* [] *name* gives *name*.

Uses Somewhat faster than *variant* if term variables are already destroyed, and their names and types are directly accessible.

See Also *variant*

SML
val **STRING** : *TYPE*;

Description This is the HOL type of strings, a type abbreviation for lists of objects of type *CHAR*.

Definition
val STRING = ⌜:*CHAR LIST*⌝;

See Also Theory "char".

SML
val **strip_app** : *TERM* −> *TERM* ∗ *TERM list*;

Description Splits a term into a head term, that is not an application, and the list of argument terms, if any, to which that head term was applied.

Example
strip_app ⌜*t t1 t2 t3* ...⌝ = (⌜*t*⌝, [⌜*t1*⌝,⌜*t2*⌝,⌜*t3*⌝,...])
strip_app ⌜*T*⌝ = (⌜*T*⌝,[])

SML
```
val strip_binder : string -> TERM -> TERM list * TERM;
```

Description *strip_binder binder* applied to

⌜*binder*(λ*vs_1*•*binder*(λ*vs_2*•...•*body*)...)⌝

will return

[⌜*vs_1*⌝,⌜*vs_2*⌝,...],⌜*body*⌝

where the *vs_i* are allowed variable structures. The function acts as *dest_binder* (q.v), and will handle paired abstraction terms. It will return an empty list and the original term if the supplied term is not formed using the binder.

See Also *strip_simple_binder*

SML
```
val strip_bin_op : string -> TERM -> TERM list
```

Description This function strips a binary operator, attempting to destroy its term argument, and recursively stripping to the right, as if by *dest_bin_op*. A term not formed from the operator is returned unchanged, as a singleton list.

Example
```
strip_bin_op "∧" ⌜a ∧ (b ∧ c) ∧ d⌝ = [⌜a⌝, ⌜b ∧ c⌝, ⌜d⌝]
```

SML
```
val strip_leaves : ('a -> 'a * 'a) -> 'a -> 'a list;
```

Description Given a function that destroys an object into a pair of objects (and here we are thinking of, for example, *dest_∧*), recursively descend the results of destruction down both branches, destroying until failure.

Example
```
strip_leaves dest_∧ ⌜(a ∧ b) ∧ c ∧ d⌝ =
        [⌜a⌝, ⌜b⌝, ⌜c⌝, ⌜d⌝]
```

SML
```
val strip_let : TERM -> ((TERM * TERM)list)list * TERM
```

Description This destroys a sequence of nested *let* constructs.

Example
```
strip_let ⌜let x = 1 in let y = 2 in x+y⌝ =
        ([[(⌜x⌝,⌜1⌝)],[(⌜y⌝,⌜2⌝)]],⌜x+y⌝)
```

SML
```
val strip_simple_binder : string -> TERM -> TERM list * TERM;
```

Description *strip_simple_binder binder* applied to

⌜*binder*(λ*v_1*•*binder*(λ*v_2*•...•*body*)...)⌝

will return

[⌜*v_1*⌝,⌜*v_2*⌝,...],⌜*body*⌝

where the *v_i* are simple variables. The function acts as *dest_simple_binder* (q.v), and will not handle paired abstraction terms. It will return an empty list and the original term if the supplied term is not formed using the binder.

See Also *strip_binder*

SML

val **strip_simple_∀** : $TERM\ {-}{>}\ (TERM\ list * TERM)$;

Description Strip a multiply universally simply quantified term.

Definition

$strip\_simple\_\forall\ \ulcorner\forall\ a\ b\ c\ ...\bullet\ body\urcorner = [\ulcorner a\urcorner,\ulcorner b\urcorner,\ulcorner c\urcorner,...],\ \ulcorner body\urcorner$

SML

val **strip_simple_∃** : $TERM\ {-}{>}\ (TERM\ list * TERM)$;

Description Strip a repeatedly existentially simply quantified term.

Definition

$strip\_simple\_\exists\ \ulcorner\exists\ a\ b\ c\ ...\bullet\ body\urcorner = [\ulcorner a\urcorner,\ulcorner b\urcorner,\ulcorner c\urcorner,...],\ \ulcorner body\urcorner$

SML

val **strip_spine_left** : $('a\ {-}{>}\ 'a * 'a)\ {-}{>}\ 'a\ {-}{>}\ 'a\ list$;

Description Given a function that destroys an object into a pair of objects (and here we are thinking of, for example, $dest\_\wedge$), recursively descend the left results of destruction, destroying until failure.

Example

$strip\_spine\_left\ dest\_\wedge\ \ulcorner(a \wedge b) \wedge c \wedge d\urcorner =$
$\quad [\ulcorner a\urcorner,\ \ulcorner b\urcorner,\ \ulcorner c \wedge d\urcorner]$

SML

val **strip_spine_right** : $('a\ {-}{>}\ 'a * 'a)\ {-}{>}\ 'a\ {-}{>}\ 'a\ list$;

Description Given a function that destroys an object into a pair of objects (and here we are thinking of, for example, $dest\_\wedge$), recursively descend the right results of destruction, destroying until failure.

See Also $strip\_bin\_op$ for stripping terms formed by binary (constant) term operators.

Example

$strip\_spine\_left\ dest\_\wedge\ \ulcorner(a \wedge b) \wedge c \wedge d\urcorner =$
$strip\_\wedge\ \ulcorner(a \wedge b) \wedge c \wedge d\urcorner =$
$\quad [\ulcorner a \wedge b\urcorner,\ \ulcorner c\urcorner,\ \ulcorner d\urcorner]$

SML

val **strip_→_type** : $TYPE\ {-}{>}\ TYPE\ list$;

Description Strip the type of a multi-argument function into its constituent types, only descending into the right hand result of $dest\_\rightarrow\_type$.

Definition

$strip\_\rightarrow\_type\ \ulcorner{:}ty1 \rightarrow ... \rightarrow tyn\urcorner =$
$\quad [\ulcorner{:}ty1\urcorner,...,\ulcorner tyn\urcorner]$

SML

val **strip_∧** : $TERM\ {-}{>}\ TERM\ list$

Description Break a term into its constituent conjuncts, descending recursively only to the right.

Example

$strip\_\wedge\ \ulcorner a \wedge (b \wedge c) \wedge d\urcorner = [\ \ulcorner a\urcorner,\ulcorner b \wedge c\urcorner,\ \ulcorner d\urcorner]$

SML

```
val strip_∨ : TERM −> TERM list
```

Description Break a term into its constituent disjuncts, descending recursively only to the right.

Example

```
strip_∨ ⌜a ∨ (b ∨ c) ∨ d⌝ = [⌜a⌝,⌜b ∨ c⌝, ⌜d⌝]
```

SML

```
val strip_⇒ : TERM −> TERM list;
```

Description Strip a multiple implication into a list of antecedents appended to the singleton list of the innermost consequent.

Definition

```
strip_⇒ ⌜t1 ⇒ t2 ⇒ ... ⇒ tn⌝ = [⌜t1⌝,⌜t2⌝,...⌜tn⌝]
```

Note that stripping a non-boolean will result in a singleton list containing that term, not a fail.

SML

```
val strip_∀ : TERM −> (TERM list * TERM);
```

Description Strip a multiply universally quantified term (perhaps with paired abstractions).

Definition

```
strip_∀ ⌜∀ a b c ...• body⌝ = [⌜a⌝,⌜b⌝,⌜c⌝,...], ⌜body⌝
```

SML

```
val strip_∃ : TERM −> (TERM list * TERM);
```

Description Strip a repeatedly existentially quantified term, possibly formed with paired abstractions.

Definition

```
strip_∃ ⌜∃ a b c ...• body⌝ = [⌜a⌝,⌜b⌝,⌜c⌝,...], ⌜body⌝
```

SML

```
val strip_ε : TERM −> (TERM list * TERM);
```

Description Strip multiple ϵ's.

Definition

```
strip_ε ⌜ε a b c ...• body⌝ = [⌜a⌝,⌜b⌝,⌜c⌝,...], ⌜body⌝
```

SML

```
val strip_λ : TERM −> (TERM list * TERM);
```

Description Strip a multiple λ-abstraction.

Definition

```
strip_λ ⌜λ a b c ...• body⌝ = [⌜a⌝,⌜b⌝,⌜c⌝,...], ⌜body⌝
```

This uses *dest_λ* (q.v.) rather than *dest_simple_λ*.

SML
```
val subst : (TERM * TERM) list -> TERM -> TERM;
```

Description *subst* $[(t\_1, u\_1), (t\_2, u\_2), \ldots]$ t returns the term formed from t by parallel substition of the t_i for the u_i. The u_i can be variables or arbitrary terms but only "free" occurrences of a u_i will be changed (i.e., only occurrences in which no free variable of u_i becomes a bound variable in t). Bound variables in t are renamed as neccessary to prevent bound variable capture.

If some u_i appears more than once in the substitution list, say $u\_i = u\_j$ for $i < j$, then the later pair (t_j, u_j) is ignored.

subst does not perform type instantiation: each t_i must have the same type as the corresponding u_i..

Definition

$subst\ [(\ulcorner t_1 \urcorner, \ulcorner u_1 \urcorner), (\ulcorner t_2, \ulcorner u_2 \urcorner),\ ...]\ \ulcorner t \urcorner = \ulcorner t[t_1/u_1,\ t_2/u_2,\ ...] \urcorner$

See Also *var_subst*

Errors

3012 ?0 and ?1 do not have the same types

SML
```
val term_any : (TERM -> bool) -> TERM -> bool;
```

Description Given a predicate on terms, tests to see if any sub-term of some term (or the term itself) satisfies the predicate. The search ceases on the first satisfaction, rather than all the tests being done and the results combined.

SML
```
val term_consts : TERM -> (string * TYPE) list;
```

Description This function extracts the subterms of a term which are constants, giving destroyed constants in each case (duplicates are eliminated)

SML
```
val term_diff : (TERM list * TERM list) -> TERM list;
```

Description Remove any terms in the first list that are α-convertible to any in the second. An infix function.

SML
```
val term_fail : string -> int -> TERM list -> 'a;
```

Description *term_fail area msg tml* first creates a list of functions from *unit* to *string*, using *string_of_term* (q.v.) providing displays of the list of terms. It then calls *fail* with the *area*, *msg* and this list of functions. This allows terms to be presented in error messages.

SML
```
val term_fold : ((TERM list) -> (TERM * 'a) -> 'a) -> (TERM * 'a) -> 'a;
```

Description *term_fold tmfun* (*tm*, *e*) traverses *tm* (depth first) and folds *tmfun* on the subterms for which it does not fail. *term_fold* does not traverse a subterm on which *tmfun* did not fail. *tmfun* has as its first argument a list giving the bound variables which are in scope at the point of use. It does not attempt to apply *tmfun* to a bound variable of an abstraction.

SML
```
val term_grab : (TERM list * TERM) -> TERM list;
```

Description If the given term is not α-convertible to any member of the list, then add it to the list. An infix function.

SML
```
val term_less : (TERM list * TERM) -> TERM list;
```

Description Remove any terms in the list that are α-convertible to the given term. An infix function.

SML
```
val term_map : ((TERM list) -> TERM -> TERM) -> TERM -> TERM;
```

Description *term_map tmfun tm* traverses *tm* (breadth first) looking for subterms for which the application *tmfun tm* does not fail and replaces such subterms with *tmfun tm*. It does not traverse the resulting subterms. *tmfun* has as its first argument a list giving the bound variables which are in scope at the point of use. It does not attempt to apply *tmfun* to a bound variable of an abstraction.

SML
```
val term_match : TERM -> TERM -> (TYPE * TYPE) list * (TERM * TERM)list;
```

Description *term_match tm_1 tm_2* attempts to find if *tm_1* is an instance of *tm_2*, up to α-convertibility. If so, then it returns two lists. The first gives the correspondence between types in *tm_1* with type variables in *tm_2*. The second gives the correspondence between (type instantiated) terms in *tm_1* with free variables in *tm_2*. Trivial (i.e. (x, x)) correspondences are not noted.

Errors
```
3054   ?0 is not a term instance of ?1
```

SML
```
val term_mem : (TERM * TERM list) -> bool;
```

Description Is the given term α-convertible to any term in the list? An infix function.

SML
```
val term_tycons : TERM -> (string * int) list;
```

Description Returns the set of type constructors and their arity present in types present within a term (represented as a list).

SML
```
val term_types : TERM -> TYPE list;
```

Description Gives a list of all the types of constants, variables or λ-abstraction variables within the term argument.

SML
```
val term_tyvars : TERM -> string list;
```

Description Returns the list of type variable names present in types present within a term.

SML
```
val term_union : (TERM list * TERM list) -> TERM list;
```

Description Take the union of two term lists viewed as sets, removing any α-convertible duplicates. An infix function.

See Also *union* for precise ordering of result.

SML
```
val term_vars : TERM -> (string * TYPE) list;
```

Description This function extracts the subterms of a term which are variables (including abstraction variables), giving destroyed variables in each case.

SML

```
val type_any : (TYPE -> bool) -> TYPE -> bool;
```

Description Given a predicate on types, tests to see if any sub-type of some type (or the type itself) satisfies the predicate. The search ceases on the first satisfaction, rather than all the tests being done and the results combined.

SML

```
val type_fail : string -> int -> TYPE list -> 'a;
```

Description *type_fail area msg tyl* first creates a list of functions from *unit* to *string*, using *string_of_type* (q.v.) providing displays of the list of types. It then calls *fail* with the *area*, *msg* and this list of functions. This allows types to be presented in error messages.

SML

```
val type_map : (TYPE -> TYPE) -> TYPE -> TYPE;
```

Description *type_map tyfun ty* traverses *ty* (breadth first) looking for subtypes, *st*, for which the application *tyfun st* does not fail and replaces such subtypes with *tyfun st*. It does not traverse the resulting subtypes.

SML

```
val type_match1 : (TYPE * TYPE) list -> TYPE -> TYPE -> (TYPE * TYPE)list;
```

Description *type_match1* is similar to *type_match*, q.v., but has an additional context parameter representing an instantiation; *type_match1* will fail unless the supplied context can be extended to give the required match. For example, the first line below evaluates true, but the second fails.

```
type_match1[(⌜:'b⌝, ⌜:'b⌝)] ⌜:('a → ℕ) → 'b⌝ ⌜:'a → 'b⌝ = [(⌜:'a → ℕ⌝, ⌜:'a⌝), (⌜:'b⌝, ⌜:'b⌝)];
type_match1[(⌜:'b → ℕ⌝, ⌜:'a⌝)] ⌜:('a → ℕ) → 'b⌝ ⌜:'a → 'b⌝;
```

Trivial associations are included in the result so that they can be passed as the context in subsequent calls. The second element of each pair in the context must be a type variable.

See Also *type_match*

Errors

```
3055 ?0 is not a type instance of ?1 in the supplied context
3019 ?0 is not a type variable
```

SML

```
val type_match : TYPE -> TYPE -> (TYPE * TYPE)list;
```

Description *type_match ty_1 ty_2* attempts to match *ty_1* with *ty_2*, i.e., to determine if *ty_1* can be obtained from *ty_2* by instantiating type variables. If so, it returns a representation of the type instantation as an association list suitable for use as an argument to *inst_type* q.v. Trivial (i.e. (x, x)) associations are not included. For example:

```
type_match ⌜:('a → ℕ) → 'b⌝ ⌜:'a → 'b⌝ = [(⌜:'a → ℕ⌝, ⌜:'a⌝)];
```

See Also *type_match1*, *inst_type*

Errors

```
3053 ?0 is not a type instance of ?1
```

SML

```
val type_of : TERM -> TYPE;
```

Description This gives the HOL type of a term.

SML
```
val type_tycons : TYPE -> (string * int) list;
```

Description This returns a list of names of type constructors, and the arity of their use, within a type.

SML
```
val type_tyvars : TYPE -> string list;
```

Description Returns the list of type variable names present in a type.

SML
```
val variant : TERM list -> TERM -> TERM;
```

Description *variant stoplist v* returns a variant of variable *v* whose name is not used for any variable in *stoplist* (which must be only variables). The variants are generated by sufficient appending of the variant string (see *set_variant_string*).

Errors
```
3007   ?0 is not a term variable
```

See Also *string_variant*, *list_variant*

SML
```
val var_subst : (TERM * TERM) list -> TERM -> TERM;
```

Description *var_subst alist term* returns the term formed by, for each pair in *alist*, substituting in *term* all free instances of the term variable which is the second of the pair with the first of the pair. The pair of the first matching term variable in the list will be used, duplicates later in the list will be ignored. Renaming may occur to prevent bound variable capture.

Note that the term variables must have the same types as the terms that are to replace them.

Definition
```
var_subst [(⌜t1⌝,⌜x1⌝),(⌜t2⌝,⌜x2⌝),...] ⌜t⌝ =
        ⌜t[t1/x1, t2/x2,...]⌝
```

Errors
```
3007   ?0 is not a term variable
3012   ?0 and ?1 do not have the same types
```

See Also *subst*

SML
```
val ~=$ : (TERM * TERM) -> bool;
```

Description An infix equality test that returns true only when its two term arguments are α-convertible, and false otherwise: no exceptions can be raised. Equality of terms is gained by using =$

SML
```
val ℕ : TYPE;
```

Description This is the HOL type of the natural numbers, $0, 1, \ldots$.

Definition
```
val ℕ = ⌜:ℕ⌝;
```

See Also Theory "ℕ".

5.2 Discrimination Nets

SML
```
signature NetTools = sig
```

Description This provides the discrimination net tools that will be used to maintain and use databases of values indexed by term form.

SML
```
type 'a NET;
```

Description This is the type of a discrimination net, its type parameter being the type of values that are handled by the net.

SML
```
val empty_net : 'a NET;
```

Description This is the starting discrimination net, which returns an empty list of values, regardless of term form.

SML
```
val list_net_enter : (TERM * 'a) list -> ('a NET) -> ('a NET);
```

Description This enters a list of values and indexing terms into a discrimination net, returning the resulting net.

SML
```
val make_net : (TERM * 'a) list -> ('a NET);
```

Description This enters a list of values and indexing terms into an empty discrimination net, returning the resulting net.

SML
```
val net_enter : (TERM * 'a) -> ('a NET) -> ('a NET);
```

Description This enters a value and its indexing term into a discrimination net, returning the resulting net.

SML
```
val net_lookup : ('a NET) -> TERM -> ('a list);
```

Description *net_lookup net term* will return a list of **at least** all the values entered into *net* that were indexed by terms which can be matched (by *term_match*, q.v.) to *term*. I.e. *term* can be produced by type and term variable instantiation from the indexing term.

A principal purpose of *net_lookup* is to make the process of rewriting a term using a list of equations and conversions more efficient by quickly filtering out items which are not applicable. Consequently speed is more important than accuracy: to use the wrong metaphor, it is not important if some inapplicable equations "slip through the net" provided all the applicable ones do as well.

The discrimination net actually returns all values whose indexing terms have the same structure as the term matched, ignoring types and variables. Thus only the pattern of constant names, combinations and abstractions will be considered, with variables in the indexing term being presumed to match any term form, regardless of type.

If *net_lookup* returns more than one value, then the only ordering on the resulting values specified is that if two entries are made into the net with the same index term, then if the *net_lookup* term matches the index term then the second entered value will be returned before the first in the list of matches.

5.3 Higher-order Matching

SML
```
val simple_ho_match : TERM -> TERM ->
        (TYPE * TYPE) list * (TERM * TERM)list;
```

Description *simple_ho_match* implements the higher-order matching algorithm of Miller and Nipkow. *term_match t p* attempts to find type and term instantiations under which p becomes $\alpha\beta\eta$-equivalent to t. The algorithm is a decision procedure for this problem in the case when p is a linear pattern, i.e., modulo, η-equivalence, p is contains no β-redexes and for each free variable v of p there is an n such that each occurrence of v is in a subterm of the form $v\,x_1 \ldots x_n$ where the *x_i* are distinct bound variables and where this subterm does not itself occur as the function in a function application (i.e., it is either the argument of a function application or the body of a λ-abstraction). The implementation does not check that p is linear pattern but will always fail rather than produce a pair of instantiations that do not make p $\alpha\beta\eta$-equivalent to t.

For example,

```
simple_ho_match ⌜∀x• x < 1⌝ ⌜∀x:'a• P x⌝ =
        ([(⌜:ℕ⌝, ⌜:'a⌝)], [(⌜λ x• x < 1⌝, ⌜P:ℕ → BOOL⌝)])
simple_ho_match ⌜∀x• x < 1⌝ ⌜∀x:'a• P x y⌝ =
        ([(⌜:ℕ⌝, ⌜:'a⌝)], [(⌜λ x (y:'b)• x < 1⌝, ⌜P:ℕ → 'b → BOOL⌝)])
```

Trivial associations (i.e. (x, x)) are omitted from the association lists returned by *ho_match*.

This function is "simple" because although it does work with paired abstractions, it cannot match terms where the pairing structure is not identical: e.g., it can match the pattern $\lambda(x,\ y)\bullet\ G\ x\ y$ to the target $\lambda(m,\ n)\bullet\ m + n$, it cannot match it to $\lambda p\bullet\ Fst\ p$.

Errors
```
120001 ?0 is not a higher-order instance of ?1
```

THE MANAGEMENT OF THEORIES AND THEOREMS

6.1 Standard ML Type Definitions

SML

```
datatype THEORY_STATUS =
        TSNormal | TSLocked | TSAncestor | TSDeleted;
```

Description Objects of this datatype indicate the status of a theory within a hierarchy, being:

Constructor	Description
TSNormal	Theory is present and may be written to.
TSLocked	Theory is present, and cannot be written to as it is locked.
TSAncestor	Theory is present, and cannot be written to as it is in an ancestor for some hierarchy.
TSDeleted	Theory has been deleted: the theory name may be reused for a new theory.

SML

```
datatype USER_DATUM =
        UD_Term of TERM * (USER_DATUM list)
|       UD_Type of TYPE * (USER_DATUM list)
|       UD_String of string * (USER_DATUM list)
|       UD_Int of int * (USER_DATUM list);
```

Description This provides a monomorphic type of trees whose nodes are labelled by terms, types, strings or integers.

Uses This type is used in the type *USER_DATA*, and may be used elsewhere, as a means of storing data that may be represented in a "reasonably general" structure for ProofPower related purposes, which also is not polymorphic.

SML

```
type CONV;
```

Description This is the type name conventionally used for conversions, that is, inference rules whose last argument is a term, and whose result is an equation whose LHS is precisely that term (no α-conversion). Though it would be type correct, we conventionally do not use this type name for other functions of type ... − > *TERM* − > *THM*.

Definition

```
type CONV = TERM -> THM;
```

SML

```
type SEQ;
```

Description This is the type of sequents, consisting of a list of assumptions and a conclusion.

Definition

```
type SEQ = (TERM list) * TERM;
```

=#provides a strict equality test on sequents, ∼=#provides an equality test on the sequents up to α-convertibility and order of assumptions.

SML
|*type* **THEORY_INFO**;

Description This is a labelled record type containing certain information associated with a theory.

Label	Type	Description
status	*THEORY_STATUS*	Current status of the theory.
inscope	*bool*	True if the theory is currently in scope (i.e. can its theorems, types and constants be usefully referred to).
contents	*THEORY*	The theory contents.
children	*int list*	List of the immediate children of the theory.
name	*string*	The name of the theory, as a string.

SML
|*type* **THEORY**;

Description A theory is a named collection of type names, constant names, axioms, definitions and theorems. In the abstract data type of theorems, the "names" of theories are represented as integers. For each type name the arity of the type is recorded and for each constant name its type is recorded. In order to allow deletion of types, constants, axioms and definitions. So-called level numbers are used to enables theorems that may depend on deleted material to be identified and rejected. In order for non-critical information such as operator fixity to be stored, a theory also includes a user-data slot which may be used to encode such information.

A theory is represented as a labelled record type, as follows:

Label	Type	Description
name	*int*	Internal representation of theory name.
ty_env	$\{arity : int, level : int\}$ *OE_DICT*	A dictionary indexed by type constructor names, returning arity, and definition level.
con_env	$\{ty : TYPE, level : int\}$ *OE_DICT*	A dictionary indexed by constant name, returning the type and definition level.
parents	*int list*	Internal representations of names of parents of theory.
del_levels	$(int * int)$ *list*	A list of ranges of deleted definition levels — if empty then no levels have been deleted.
axiom_ dict	*THM OE_DICT*	A dictionary of axioms.
defn_dict	*THM OE_DICT*	A dictionary of definitions.
thm_dict	*THM OE_DICT*	A dictionary of theorems.
current_ level	*int*	The current definition level.
user_data	*USER_DATA ref*	The user data stored in the theory.

SML
|*type* **THM**;

Description This is the abstract data type of theorems in ProofPower, whose primitive constructors are the inference rules and extensional mechanisms of the abstract data type. =|−provides a strict equality test on the conclusion and assumptions of theorems, ∼=|−provides an equality test on the conclusion and assumptions of theorems up to α-convertibility and order of assumptions.

SML
```
type USER_DATA;
```

Description This is the type of a store for objects of type *USER_DATUM*. It is implemented as:

ML
```
type USER_DATA = USER_DATUM S_DICT;
```

Uses Within the type *THEORY* it is used to include such details as the fixity of types and constants.

6.2 Symbol Table

SML
```
signature SymbolTable = sig
```

Description This is the signature for the structure which contains the symbol table and its access functions. This structure contains private functions which are invoked as one navigates around the theory database. These private functions may give rise to error 20001 if the theory database user data has been corrupted (e.g. by explicit and incorrect use of the lower level interfaces).

Any of the functions in the structure which update the current theory may give rise to error 20002

Errors
```
20001 A symbol table entry in theory ?0 is corrupt (use restore_defaults to clear)
20002 The current theory, ?0, is not open for writing
20003 Internal error: ?0
```

SML
```
val declare_alias : (string * TERM) -> unit;
```

Description *declare_alias* (s, c) declares s as an alias for the constant c. s must comply with the HOL lexical rules for an identifier.

Errors
```
20301 The term ?0 is not a constant
20302 The string ?0 is already in use as an alias for ?1
20305 The constant ?0 is not in scope
20306 The string ?0 is not an identifier
```

SML
```
val declare_binder : string -> unit;
```

Description *declare_binder* s declares s to have the syntactic status of a binder in the current context. s must comply with the HOL lexical rules for an identifier and must not be the string ",".

See Also *undeclare_fixity*

Errors
```
20201 A fixity declaration is not allowed for ?0 (which is not an identifier)
20202 Cannot change the fixity of ‘,‘
```

SML
```
val declare_const_language : string * string -> unit;
```

Description *declare_const_language* (s, l) adds the language indicator l to those associated with the name s when used as a constant in the current context.

Errors
```
20501 There is no constant called ?0 in the current context
```

SML
```
val declare_left_infix : (int * string)-> unit;
val declare_right_infix : (int * string)-> unit;
val declare_infix : (int * string)-> unit;
```

Description *declare_left_infix* (p, s) declares s to have the syntactic status of an left associative infix operator with precedence p in the current context. s must comply with the HOL lexical rules for an identifier.

Similarly, *declare_right_infix* is used to declare right associative operators. *declare_infix* is provided for compatibility with earlier versions of the system and is the same as *declare_right_infix*.

See Also *undeclare_fixity*

Errors
20201 A fixity declaration is not allowed for ?0 (which is not an identifier)

SML
```
val declare_nonfix : string -> unit;
```

Description *declare_nonfix s* undoes the effect of a declaration of s to have special syntactic status (using *declare_binder*, *declare_infix*, *declare_prefix* or *declare_postfix*).

The effect of *declare_nonfix s* depends on the theory in which the special status for s was declared: if it was declared in the current theory, then the declaration is just removed; if in an ancestor theory then a declaration for s as a nonfix is inserted in the current theory. (Thus in the first case, the syntactic status for s reverts to what it was before the earlier declaration, whereas in the second case the syntactic status will be suppressed.)

s must must not be the string ",".

See Also *undeclare_fixity*

Errors
20201 A fixity declaration is not allowed for ?0 (which is not an identifier)
20202 Cannot change the fixity of ',‘
20203 There is no fixity declaration for ?0 in the current context

SML
```
val declare_postfix : (int * string) -> unit;
```

Description *declare_postfix* (p, s) declares s to have the syntactic status of a postfix operator with precedence p in the current context. s must comply with the HOL lexical rules for an identifier and must not be the string ",".

See Also *undeclare_fixity*

Errors
20201 A fixity declaration is not allowed for ?0 (which is not an identifier)
20202 Cannot change the fixity of ',‘

SML

val **declare_prefix** : (*int* ∗ *string*) −> *unit*;

Description *declare_prefix* (p, s) declares s to have the syntactic status of a prefix operator with precedence p in the current context. s must comply with the HOL lexical rules for an identifier and must not be the string ",".

See Also *undeclare_fixity*

Errors

20201 A fixity declaration is not allowed for ?0 (which is not an identifier)
20202 Cannot change the fixity of ','

SML

val **declare_terminator** : *string* −> *unit*

Description *declare_terminator* s checks that s is a valid terminator, and if so declares that s is to be used as a lexical terminator in the current context.

Errors

20101 The string ?0 is not a valid terminator. Terminators must start
with a symbolic character, must not contain spaces,
and must not end with underscore, ⅄ or Y
20102 The string ?0 is already declared as a terminator

SML

val **declare_type_abbrev** : (*string* ∗ *string list* ∗ *TYPE*) −> *unit*;

Description *declare_type_abbrev* $(s, [\alpha\_1, \ldots, \alpha\_k], \tau)$ declares $(\alpha\_1, \ldots, \alpha\_k)s$ as a type abbreviation for the type τ. The identifier s may not already have been declared as a type abbreviation or be the name of a type constructor defined in the present context, in which cases a warning message is issued. s must comply with the HOL lexical rules for an identifier.

Errors

20401 The identifier ?0 is already declared as a type abbreviation
20402 The identifier ?0 is already declared as a type constructor
20407 The formal parameter list ?0 contains duplicate type variable names
20408 The string ?0 is not an identifier

SML

val **expand_type_abbrev** : (*string* ∗ *TYPE list*) −> *TYPE*;

Description *expand_type_abbrev* $s, [\tau\_1, \ldots, \tau\_k]$ is the expansion of the type abbreviation s with respect to the arguments $[\tau\_1, \ldots, \tau\_k]$.

Errors

20404 The identifier ?0 is not declared as a type abbreviation
20405 The type abbreviation ?0 should have ?1 argument not ?2
20406 The type abbreviation ?0 should have ?1 arguments not ?2

SML

val **get_aliases** : *string* −> (*string* ∗ *TERM*) *list*;

Description *get_aliases thy* returns information about identifiers which have have been declared as aliases in the theory *thy*. The return value is a list of pairs. Each pair contains a name and a constant for which that name is an alias. The same name may be used as an alias for several different constants, and if this happens there will be multiple entries for that alias in the list.

Errors

20601 There is no theory called ?0

SML

| *val* **get_alias_info** : *string* −> (*string* ∗ *TYPE*)*list OPT*;

Description *get_alias_info c* returns the list of aliases for the constant with name c, or *Nil* if c is not the name of a constant. For each pair (a, τ) in the result, a is an alias for c at instances of the type τ.

SML

| *val* **get_alias** : (*string* ∗ *TYPE*) −> *string*;

Description *get_alias*(c, τ) returns the most appropriate alias for the constant with name c at the type τ. If no aliases for the name c have been declared then c is returned otherwise the most recent alias s associated with a type τ' which can be instantiated to τ is returned.

SML

| *val* **get_binders** : *string* −> *string list*;

Description *get_binders thy* returns the list of identifiers which have been declared as binders in the theory *thy*.

Errors

| *20601 There is no theory called ?0*

SML

| *val* **get_const_info** : *string* −> (*TYPE* ∗ ((*string* ∗ *TYPE*)*list*)) *OPT*;

Description *get_const_info a* returns the information, (τ, cs), associated with the name a used as a constant name or an alias for a constant, if any. cs is the list of names and types of constants to which a might refer (as an alias or as the actual constant name). τ is the type to use for this name during type inference, namely, the antiunifier of the types in cs.

SML

| *val* **get_const_language** : *string* −> *string list*;

Description *get_const_language s* returns the language indicators associated with the name s when used as a constant in the current context. If there is no constant called s, then *get_const_language s* returns the language indicator associated with the current theory. The language indicator is "*HOL*" for all identifiers supplied as part of the ICL HOL system. The head element of the list returned is the language indicator associated with the constant's declaring theory.

SML

| *val* **get_current_language** : *unit* −> *string*;

Description *get_current_language* () returns the language indicator associated with the current theory.

SML

| *val* **get_current_terminators** : *unit* −> *string list list*;

Description *get_current_terminators*() returns the list of identifiers which have been declared as terminators in the current context using *new_terminator*. The names are returned in exploded form, i.e. as a list of strings each containing one character.

SML

| *val* **get_fixity** : *string* −> *Lex.FIXITY*;

Description *get_fixity s* returns the syntactic status of s in the current context.

SML
```
val get_language : string -> string;
```

Description *get_language thy* returns the language indicator associated with the theory *thy*.

Errors

20601 There is no theory called ?0

SML
```
val get_left_infixes : string -> (int * string) list;
val get_right_infixes : string -> (int * string) list;
```

Description *get_left_infixes thy* (resp. *get_right_infixes thy*) returns the list of identifiers (and associated precedences) which have been declared as left (resp. right) associative infix operators in the theory *thy*.

Errors

20601 There is no theory called ?0

SML
```
val get_nonfixes : string -> string list;
```

Description *get_nonfixes thy* returns the list of identifiers which are declared as binder, infix, prefix or postfix in an ancestor of the theory *thy*, but have had that special status suppressed (using *declare_nonfix*) in the theory *thy* itself.

Errors

20601 There is no theory called ?0

SML
```
val get_postfixes : string -> (int * string) list;
```

Description *get_postfixes thy* returns the list of identifiers (and associated precedences) which have been declared as postfix operators in the theory *thy*.

Errors

20601 There is no theory called ?0

SML
```
val get_prefixes : string -> (int * string) list;
```

Description *get_prefixes thy* returns the list of identifiers (and associated precedences) which have been declared as prefix operators in the theory *thy*.

Errors

20601 There is no theory called ?0

SML
```
val get_terminators : string -> string list;
```

Description *get_terminators thy* returns the list of identifiers which have been declared as terminators in the theory *thy*.

Errors

20601 There is no theory called ?0

SML
```
val get_type_abbrev : string -> (string list * TYPE);
```

Description *get_type_abbrev s* returns the formal argument list and type associated with the type abbreviation *s*.

Errors

20404 The identifier ?0 is not declared as a type abbreviation

SML

val **get_type_abbrevs** : *string* −> (*string* ∗ (*string list* ∗ *TYPE*))*list*;

Description *get_type_abbrevs thy* returns information about the type abbreviation declarations which have been made in the theory *thy*. The return value is a list of pairs. Each pair contains the name of the corresponding type abbreviation together with its formal arguments and the type for which it is an abbreviation.

Errors

20601 There is no theory called ?0

SML

val **get_type_info** : *string* −> (*int* ∗ (*string list* ∗ *TYPE*) *OPT*) *OPT*;

Description *get_type_info s* returns the type information, if any, associated with *s*. See DS/FMU/IED/DTD020 for more information.

SML

val **get_undeclared_terminators** : *string* −> *string list*;

Description *get_undeclared_terminators thy* returns the list of identifiers whose status as terminators has been suppressed (with *undeclare_terminator*) in the theory *thy*.

Errors

20601 There is no theory called ?0

SML

val **get_undeclared_type_abbrevs** : *string* −> *string list*;

Description *get_undeclared_type_abbrevs thy* returns the list of identifiers which have have had their status as type abbreviations suppressed in the theory *thy*.

Errors

20601 There is no theory called ?0

SML

val **get_undeclared_aliases** : *string* −> (*string* ∗ *TERM*) *list*;

Description *get_undeclared_aliases thy* returns information about aliases which have been suppressed (with *undeclare_alias*) in the theory *thy*. The return value is a list of pairs. Each pair contains a name and a constant for which that name is no longer to be used as an alias. There may be more than one entry for a given name in the list (since several *undeclare_alias* commands may apply to one name).

Errors

20601 There is no theory called ?0

SML

val **is_type_abbrev** : *string* −> *bool*;

Description *is_type_abbrev s* returns *true* iff. *s* is declared as a type abbreviation

SML

val **resolve_alias** : (*string* ∗ *TYPE*) −> *TERM*;

Description *resolve_alias*(s, τ) returns a term of the form *mk_const*(c, τ) where c is the "best' resolution for the identifier s. This best resolution will be s if s has been introduced as a constant of type τ' where τ' is an instance of τ. If s is an alias then c is taken from the alias declaration for s in which the aliased constant has a type τ' which can be instantiated to τ. If more than one such declaration is applicable the most recent one is used.

Errors

20304 The identifier ?0 is not a valid constant name (or alias) at this type

SML
```
val restore_defaults : unit -> unit;
```

Description *restore_defaults*() may be used to clear corrupted symbol table information in the current theory. It does this by restoring the theory to the state it would have if no terminator, fixity, alias, type abbreviations or language declarations had been performed. A warning message is issued (and the interactive user is prompted as to whether to continue) before the operation is performed.

Errors

20703 This operation will delete all symbol table information from theory ?0

SML
```
val set_current_language : string -> unit;
```

Description *set_current_language s* sets the language indicator associated with the current theory to s.

SML
```
val undeclare_alias : (string * TERM) -> unit;
```

Description *undeclare_alias* (s, c) reverses the effect of a declaration of s as an alias for the constant c in the current context. This includes the possibility that s is the name of c itself.

The precise effect depends on the theory in which the alias was declared: if it was declared in the current theory, then the declaration is just removed (so that if s is declared as an alias for c in an ancestor theory, s will still act as an alias for c in the current theory); if in an ancestor theory then arrangements are made in the current theory to prevent s acting as an alias for c.

If s is the name of c itself, the type inferrer will no longer recognise s as a reference to c. In this case, c may be accessed either via an alias or via an ML quotation. This gives a work-around for the potential problem when a theory contains a constant whose name is needed as a variable name in some application using the theory.

Errors

20301 The term ?0 is not a constant
20303 The identifier ?0 is not declared as an alias for ?1

SML
```
val undeclare_terminator : string -> unit
```

Description *undeclare_terminator s* removes s from the list of identifiers which act as terminators for parsing purposes in the current context.

Errors

20103 ?0 is not in the list of terminators in the current context

SML
```
val undeclare_type_abbrev : string -> unit;
```

Description *undeclare_type_abbrev* $(s, [\alpha\_1, \ldots, \alpha\_k], \tau)$ reverses the effect of a declaration of s as a type abbreviation.

The precise effect depends on the theory in which the type abbreviation was declared: if it was declared in the current theory, then the declaration is just removed (so that if s is declared as a type abbreviation in an ancestor theory, s will revert to whatever that declaration said); if in an ancestor theory then arrangements are made in the current theory to prevent s being treated as a type abbreviation.

Errors

20403 The identifier ?0 is not declared as a type abbreviation

6.3 The Kernel Interface

SML

```
signature KernelInterface = sig
```

Description This is the signature of the structure that gives the standard interface to the logical kernel. This interface adds a layer of additional services to the kernel functionality. E.g., it is used to notify the parser and type-inferrer so that they operate correctly when the current theory changes. The functions in the structure *KernrlInterface* should always be used in preference to direct use of the functions in the structure *pp'Kernel* except in coding extensions to the system that need to bypass these services.

Errors

```
6013 ?0 is ill-formed in current theory: type name ?1 is not declared
6014 ?0 is ill-formed in current theory: type name ?1 does not have arity used
6015 ?0 is ill-formed in current theory: constant name ?1 not declared
6038 ?0 is ill-formed in current theory: constant name ?1 cannot have type used
```

The above are error messages various kinds of well-formedness check failures. A well-formedness check occurs on any types, terms and theorems saved in a theory, and thus these errors may occur for any function in this signature which saves types, terms or theorems in a theory.

SML

```
datatype KERNEL_INFERENCE =
            KISubstRule of (THM * TERM) list * TERM * THM * THM
        |   KISimpleλEqRule of TERM * THM * THM
        |   KIInstTypeRule of (TYPE * TYPE) list * THM * THM
        |   KI⇒Intro of TERM * THM * THM
        |   KI⇒Elim of THM * THM * THM
        |   KIAsmRule of TERM * THM
        |   KIReflConv of TERM * THM
        |   KISimpleβConv of TERM * THM
        |   KISucConv of TERM * THM
        |   KIStringConv of TERM * THM
        |   KICharConv of TERM * THM
        |   KIEqSymRule of THM * THM
        |   KIListSimple∀Elim of TERM list * THM * THM
        |   KIEqTransRule of THM * THM * THM
        |   KIMkAppRule of THM * THM * THM
        |   KI⇔MPRule of THM * THM * THM
        |   KISimple∀Intro of TERM * THM * THM
        |   KIInstTermRule of (TERM * TERM) list * THM * THM
        |   KIPlusConv of TERM * THM;

val on_kernel_inference : (KERNEL_INFERENCE -> unit) -> unit;
```

Description The call *on_kernel_inference f* registers the function *f* to be called whenever a kernel inference rule is called successfully. Several functions may be registered and they will be called in order of registration.

A value of type *KERNEL_INFERENCE* is passed to represent the instance of the rule that has been called. The tuple forming the argument to each constructor of the type gives the arguments and result of the corresponding rule.

SML

```
(∗ compactification_mask : integer control: default: compiler−dependent ∗)
val get_compactification_cache : unit −> TYPE list;
val set_compactification_cache : TYPE list −> unit;
val clear_compactification_cache : unit −> unit;
```

Description These functions and associated control value support compactification of objects stored in the theory database.

set_compactification_cache and *get_compactification_cache* may be used at the beginning and end of a ProofPower session to preserve the contents of the cache of type information which is used to implement compactification. Internally, the cache is held as a rather more complex, and much larger, data structure than a simple list of types and so *clear_compactification_cache* is used automatically to empty the cache at the end of a session, thereby avoiding saving the data structure in the database file. Restoring the cache from the list returned by *get_compactification_cache* using *set_compactification_cache* is time-consuming and is not done automatically; however, doing this using, e.g., the following lines of ML, may improve the space-saving in applications which are built up in several sessions:

SML Example - End of Every Session

```
val saved_compactification_cache = get_compactification_cache();
```

SML Example - Beginning of Second and Later Sessions

```
set_compactification_cache saved_compactification_cache;
```

ML functions which compute terms can often be coded so as to produce terms in which common subterms are shared. The compactification algorithm may actually increase the space occupied by such terms. Producers of such functions may therefore wish to suppress the compactification when the computed terms are stored in the theory database.

compactification_mask is an integer control which is treated as a bit-mask and may used to suppress selected aspects of the compactification algorithm. The default value of *0* should be correct for most normal specification and proof work. The significance of the bits in the mask is as follows:

1	Suppress compactification in *new_axiom*
2	Suppress compactification in *new_const*
4	Suppress compactification in *new_type_defn*
8	Suppress compactification in *new_spec*
16	Suppress compactification in *save_thm*
32	Suppress compactification in *simple_new_defn*

So, for example, if the mask is set to *47* (= *1* + *2* + *4* + *8* + *32*), then compactification will only be performed when *save_thm* is called. The default value depends on the Standard ML compiler: 63 (i.e., no compactification) for Poly/ML and 0 (i.e., full compactification) for Standard ML of New Jersey.

SML

datatype **KERNEL_STATE_CHANGE**
= **OpenTheory** *of string* ∗ ((*string list*) ∗ (*string list*))
| **DeleteTheory** *of string*
| **NewTheory** *of string*
| **NewParent** *of string* ∗ (*string list*)
| **LockTheory** *of string*
| **UnlockTheory** *of string*
| **DuplicateTheory** *of string* ∗ *string*
| **SaveThm** *of string* ∗ *THM*
| **ListSaveThm** *of string list* ∗ *THM*
| **DeleteConst** *of TERM*
| **DeleteType** *of string*
| **DeleteAxiom** *of string*
| **DeleteThm** *of string*
| **NewAxiom** *of* (*string list* ∗ *TERM*)∗*THM*
| **NewConst** *of string* ∗ *TYPE*
| **NewType** *of string* ∗ *int*
| **SimpleNewDefn** *of* (*string list* ∗ *string* ∗ *TERM*) ∗ *THM*
| **NewTypeDefn** *of* (*string list* ∗ *string* ∗ (*string list*) ∗ *THM*) ∗ *THM*
| **NewSpec** *of* (*string list* ∗ *int* ∗ *THM*) ∗ *THM*
| **SetUserDatum** *of string* ∗ *USER_DATUM*;

Description This is an encoding of the arguments of the functions of signature *KernelInterface* which change the state of the theory database. When used to notify the system of a change that has been made certain additional information is also included. If used to notify the system before a change is made the slots will be given "null" default values ("", [], *asm_rule mk_t*).

Operation	Value	Description
open_theory	(*thy*, (*inthys*, *outthys*))	*thy* names the theory which has been opened. *inthys* names the theories which have come into scope. *outthys* names the theories which have gone out of scope.
new_parent	(*thy*, *inthys*)	*thy* names the new parent theory. *inthys* names the theories which have come into scope.
SimpleNewDefn *NewTypeDefn* *NewSpec* *NewAxiom*	(*arg*, *thm*)	*arg* gives the argument to the operation. *thm* is the new defining theorem.

SEE ALSO *on_kernel_state_change*, *before_kernel_state_change*

SML

```
type CHECKPOINT;
val checkpoint : string -> CHECKPOINT;
val rollback : CHECKPOINT -> unit;
```

Description This opaque type and its associated functions implement a system for checkpointing and restoring the state of the theory hierarchy. It is intended primarily for programmatic use in applications that may need to undo multiple extensions to the logical contents of the theory and changes to user data. The check-pointing scheme is unable to keep track of theories, theorems, definitions etc. that have been deleted. Applications that may delete such objects must make their own arrangements for restoring the deleted objects.

The parameter to *checkpoint* is a theory name. The checkpoint returned contains the information required by *rollback* to roll the indicated theory and all its descendants back to the state it had when the checkpoint was taken. The theory becomes the current theory after the rollback.

Rolling back is done using *delete_const* etc. and so rolling back the state of definitions and axioms is restricted to changes made in theories which did not have children when the checkpoint was taken. For uniformity, *rollback* does not attempt to restore the state of the theorems and the user data in theories which had children when the checkpoint was taken. A theory that has been introduced and has become a parent of a theory that existed when the checkpoint was taken will not be deleted (otherwise the child theory would also have to be deleted).

Messages 12015 to 12017 are reported by *rollback* as comments. In general, *rollback* will just report on the problem and continue trying to restore other theories. For example, if *rollback* is unable to delete a theory, it continues to attempt to restore the state of the definitions, etc. in the theories that are to be retained. This is an unlikely situation, since *rollback* unlocks a theory if necessary before trying to delete it, so it will only happen if the application using *rollback* has created a new theory hierarchy and a theory to be deleted has obtained ancestor status. Message 12020 is reported by *rollback* as a failure.

Errors

```
12015  it was not possible to delete theory ?0
12016  the theory ?0 has been deleted since the checkpoint was taken; this change cannot
       be rolled back
12017  a failure was reported while trying to restore theory ?0 (?1)
12020  the theory ?0 has been deleted since this checkpoint was taken and a new
       theory of the same name has been created. Rolling back to this checkpoint
       is not possible.
```

SML

```
val =|- : THM * THM -> bool;
val ~=|- : THM * THM -> bool;
val =# : SEQ * SEQ -> bool;
val ~=# : SEQ * SEQ -> bool;
```

Description =|− provides a strict equality test on the conclusion and assumptions of theorems, ∼=|− provides an equality test on the conclusion and assumptions of theorems up to α-convertibility and order of assumptions. =# provides a strict equality test on sequents, ∼=# provides an equality test on the sequents up to α-convertibility and order of assumptions.

SML

```
val asms : THM -> TERM list;
```

Description This returns the assumptions(hypotheses) of a theorem.

See Also *dest_thm*

SML
```
val before_kernel_state_change : (KERNEL_STATE_CHANGE -> unit) -> unit
```

Description *before_kernel_state_change f* nominates *f* to be called before the theory database is to be modified by functions from the signature *KernelInterface*. The argument to *f* encodes the operation which caused the modification together with its arguments and certain other additional information (usually sets to null defaults for this function). A list of such functions is maintained, and the new function is put at the end of the list, which means it may, if desired undo or overwrite the effects of a function nominated by an earlier call of *before_kernel_state_change*.

Functions handled by *before_kernel_state_change* might be used to raise errors to prevent the state change occurring. This will prevent further checks or actions being made. Thus a careful choice between *before_* or *on_* is called for.

See Also *KERNEL_STATE_CHANGE*, *on_kernel_state_change*

SML
```
val compact_type : TYPE -> TYPE;
val compact_term : TERM -> TERM;
val compact_thm : THM -> THM;
```

Description These functions compactify type, term and theorem values, currently by commoning up type information so that only one ML instance of any type is used in the compactified value. Depending on the value of the integer control variable *compactification_mask*, q.v., these interfaces are invoked automatically as values are stored in the theory database.

The *compactify_$\mathcal{XXX}$* interfaces act as identify functions: *compactify_$\mathcal{XXX}$ x* returns a value which is equal to *x* (in the sense of =:, = \$ or = |− as appropriate), but which usually occupies significantly less space than *x*.

SML
```
val concl : THM -> TERM;
```

Description This returns the conclusion of a theorem.

See Also *dest_thm*

SML
```
val delete_axiom : string -> unit
```

Description *delete_axiom key* deletes the axiom stored under key *key* and any other object which depends on it from the current theory. If any objects do depend on the axiom, the interactive user will be notified and asked whether to proceed with the deletion.

After the deletion any theorems which have been proven since the introduction of the axiom will no longer be usable for further proof.

Note that the deletion will attempt to delete all necessary theorems before deleting constants, types, and axioms in single steps, and thus may fail with a partially modified theory. This is because checks in the interface may not be as definitive as those of the kernel. The "on kernel state change" functions will be notified as if all necessary single step deletions, of theorems, constants, types and axioms to achieve the goal had been done, but after all the actual changes have been made. The "before kernel state change" functions will be notified of all the changes, as if single steps, before any are made.

Errors
```
6037   Theory ?0 is locked
6071   Theory ?0 is a read-only ancestor
6076   Theory ?0 has child theories
12003  Theory ?0 does not contain an axiom under key ?1
12012  Deletion of ?0 would require the deletion of ?1
```

SML
```
val delete_const : TERM -> unit
```

Description *delete_const c* deletes the constant *c* (or the constant with the same type, up to renaming of type variables) and any other object which depends on *c* from the current theory. If *c* is the application of a constant to some arguments then that constant is the one deleted. If any saved objects other than *c* and its defining theorem do depend on *c*, the interactive user will be notified and asked whether to proceed with the deletion.

After the deletion any theorems which have been proven since the definition of *c* will no longer be usable for further proof.

Note that the deletion will attempt to delete all necessary theorems before deleting constants, types, and axioms in single steps, and thus may fail with a partially modified theory. This is because checks in the interface may not be as definitive as those of the kernel. The "on kernel state change" functions will be notified as if all necessary single step deletions, of theorems, constants, types and axioms to achieve the goal had been done, but after all the actual changes have been made. The "before kernel state change" functions will be notified of all the changes, as if single steps, before any are made.

Errors
```
6037   Theory ?0 is locked
6071   Theory ?0 is a read-only ancestor
6076   Theory ?0 has child theories
12001  Theory ?0 does not contain the constant ?1 with the supplied type
12012  Deletion of ?0 would require the deletion of ?1
12014  ?0 is not a constant or a constant applied to some arguments
```

SML
```
val delete_theory : string -> unit;
```

Description *delete_theory thy* removes the theory *thy* from the theory database. This means, for instance, that all theorems that were proven with the deleted theory as the current theory, and all constants and types declared within the theory, will become out of scope.

Errors
```
12035 Theory ?0 is not present in the current hierarchy
6037  Theory ?0 is locked
6069  Theory ?0 is in scope
6071  Theory ?0 is a read-only ancestor
6076  Theory ?0 has child theories
```

SML
```
val delete_thm : string -> THM;
```

Description *delete_thm key* deletes the theorem stored under key *key* from the current theory. It returns the deleted theorem.

Errors
```
6037  Theory ?0 is locked
6046  Key ?0 is not used for a theorem in theory ?1
6071  Theory ?0 is a read-only ancestor
```

SML
```
val delete_to_level :
        {do_warn : bool,
         caller : string,
         target : string,
         level : int} -> (string * int) list * (string * TYPE) list;
val thm_level : THM -> int;
```

Description *delete_to_level* deletes constants, types and axioms (and any theorems that may depend on them) down to a specified level number. *do_warn* specifies whether or not the user should be warned before doing this. *caller* is the name of the calling function for use in error messages. *target* is the name of the target being deleted for use in the warning message. *level* is the level of the constant, type or axiom which is the target to be deleted. The returned value comprises the lists of types and constants that have been deleted (with their arities and types).

The level numbers for constants and types may be retrieved using the data structure returned by *get_theory*. *thm_level* returns the level number associated with a theorem or axiom.

SML
```
val delete_type : string -> unit
```

Description *delete_type t* deletes the type constructor *t* and any other object which depends on *t* from the current theory. If any objects other than *t* and its defining theorem do depend on *t*, the interactive user will be notified and asked whether to proceed with the deletion.

After the deletion ny theorems which have been proven since the definition of *ty* will no longer be usable for further proof.

Note that the deletion will attempt to delete all necessary theorems before deleting constants, types, and axioms in single steps, and thus may fail with a partially modified theory. This is because checks in the interface may not be as definitive as those of the kernel. The "on kernel state change" functions will be notified as if all necessary single step deletions, of theorems, constants, types and axioms to achieve the goal had been done, but after all the actual changes have been made. The "before kernel state change" functions will be notified of all the changes, as if single steps, before any are made.

Errors
```
6037   Theory ?0 is locked
6071   Theory ?0 is a read-only ancestor
6076   Theory ?0 has child theories
12002  Theory ?0 does not contain the type constructor ?1
12012  Deletion of ?0 would require the deletion of ?1
```

SML
```
val dest_thm : THM -> SEQ;
```

Description This returns the representation of a theorem as a sequent, i.e. as a list of assumptions and a conclusion.

See Also *asms*, *concl*

SML

```
val do_in_theory : string -> ('a -> 'b) -> 'a -> 'b;
```

Description *do_in_theory thy f a* will change to the named theory *thy*, apply f to a, and return to the theory in which it was called. It will not notify the kernel state change functions (e.g. *on_kernel_state_change*) when it changes to the named theory, nor will it notify them on its return. Thus for instance the symbol table mechanism, and so term parsing, will behave as if no theory change had taken place before the application of f to a. This refusal to notify causes this function to be faster than the appropriate two uses of *open_theory*.

The function prevents the application of f from once more changing the current theory to another, or functions that may delete the original theory. The block will provoke error 12011. These functions are:

```
open_theory    new_theory    delete_theory
```

It will also discard any changes made by *before_kernel_state_change* during the application of f at its end.

The function will intercept any exceptions (including keyboard interrupts), and will attempt to remove the block on changing the current theory, and then return to the original theory. However, in certain circumstances (such as multiple keyboard interrupts, or use of pp' functions) the exception handler itself may be interrupted or be otherwise unable to complete its work. In these cases *open_theory* must be used by hand to notify the proof system of the correct theory and its context. If this raises the error *12011* then repeat the use of *open_theory*, as each raising of the error involves the removal of one block put in place by *do_in_theory* before the message is generated.

Errors

```
12011 Blocked from changing the current theory.
      This particular block has now been removed.
      Exceptionally, further blocks, giving the same
      error message, may still be in place. These blocks
      should be cleared now by repeatedly trying open_theory
      until this error message is not provoked
12013 An internal error has corrupted the current theory
      data. Immediately make a call of open_theory
      to clear this internal error
12203 The kernel interface tables were in an inconsistent state.
      The tables are now being rebuilt.
```

SML
```
val duplicate_theory : (string * string) -> unit;
```

Description *duplicate_theory oldthy newthy* creates a new theory, called *newthy* with the same contents and parents as *oldthy*, but without any children. The current theory remains unchanged.

Uses To allow the user to modify and experiment with a theory that has child theories that are not involved in the experiment, and would perhaps clash with the experiment.

Errors

6026	*Theory ?0 may not be duplicated (it must always be in the scope of any opened theory)*
6042	*Theory ?0 may not be duplicated (the duplicate would not be a descendant of ?1)*
12035	*Theory ?0 is not present in the current hierarchy*
6040	*Theory ?0 is already present in current theory hierarchy*

To ensure that the duplicate theory can be opened by *open_theory* (q.v.) the system will prevent the duplication of theories which would give rise to error 6017 of *open_theory* if opened, and attempts to create such duplicates will give rise to error 6026 or 6042.

SML
```
val get_ancestors : string -> string list;
```

Description This returns all the ancestors of the named theory, including the theory itself. The named theory is the last name in the list returned. The name of the parent first added to the named theory is next to last, preceded by its ancestors. All these are preceded by the second parent theory and its ancestors, apart from those already added. These are preceded by any unnoted ancestors of the third, fourth, etc parents of the named theory. The order in the list of the ancestors of the parent theories is determined recursively by this ordering.

Errors

12035	*Theory ?0 is not present in the current hierarchy*

SML
```
val get_axioms : string -> (string list * THM) list;
val get_axiom_dict : string -> THM OE_DICT;
```

Description *get_axioms* returns all the axioms stored in the indicated theory together with the keys under which they are stored.

get_axiom_dict returns the mapping of keys to axioms represented as an order-preserving efficient dictionary.

Errors

12035	*Theory ?0 is not present in the current hierarchy*

SML
```
val get_axiom : string -> string -> THM;
```

Description *get_axiom theory key* returns the axiom with key *key*, found in theory *theory*.

To improve performance, this function uses a cache containing the values of previous calls. This cache is rebuilt when *open_theory* is called, by removing entries that have gone out of scope. Opening a theory such as *basic_hol* that is low down in the theory hierarchy will reclaim the memory occupied by the cache.

Errors

12035	*Theory ?0 is not present in the current hierarchy*
12005	*Theory ?0 does not have an axiom with key ?1*
12010	*Theory ?0 is not in scope*

SML
```
val get_children : string -> string list;
```

Description This returns the immediate children of the named theory, (not including the theory itself).

Errors
```
12035 Theory ?0 is not present in the current hierarchy
```

SML
```
val get_consts : string -> TERM list;
```

Description This returns (most general instances of) all the constants stored in a theory.

Errors
```
12035 Theory ?0 is not present in the current hierarchy
```

SML
```
val get_const_keys : string -> E_KEY list;
```

Description This returns the efficient dictionary keys that represent the names of the constants stored in a theory.

Errors
```
12035 Theory ?0 is not present in the current hierarchy
```

SML
```
        val get_const_theory : string -> string;
```

Description *get_const_theory c* returns the name of the theory in which the constant *c* is defined.

Errors
```
12201 There is no constant called ?0 in the current context
```

SML
```
val get_const_type : string -> TYPE OPT;
```

Description If a constant with the given name is in scope, then its type is returned, otherwise *Nil*.

Uses This is likely to be often used just as a rapid test for a constant being in scope.

See Also *get_const_info*

SML
```
val get_current_theory_name : unit -> string;
```

Description Returns the name of the current theory.

SML
```
val get_current_theory_status : unit -> THEORY_STATUS;
```

Description This returns the current theory's status.

SML
```
val get_defns : string -> (string list * THM) list;
val get_defn_dict : string -> THM OE_DICT;
```

Description *get_defns* returns all the defining theorems stored in the indicated theory together with the keys under which they are stored.

get_defn_dict returns the mapping of keys to defining theorems represented as an order-preserving efficient dictionary.

Errors
```
12035 Theory ?0 is not present in the current hierarchy
```

SML
```
val get_defn : string -> string -> THM;
```

Description *get_defn theory key* returns the definition with key *key*, found in theory *theory*.

To improve performance, this function uses a cache containing the values of previous calls. This cache is rebuilt when *open_theory* is called, by removing entries that have gone out of scope. Opening a theory such as *basic_hol* that is low down in the theory hierarchy will reclaim the memory occupied by the cache.

Errors
```
12035 Theory ?0 is not present in the current hierarchy
12004 Theory ?0 does not have a definition with key ?1
12010 Theory ?0 is not in scope
```

SML
```
val get_descendants : string -> string list;
```

Description This returns all the descendants of the named theory, including itself.

Errors
```
12035 Theory ?0 is not present in the current hierarchy
```

SML
```
val get_parents : string -> string list;
```

Description This returns the immediate parents of the named theory, (not including the theory itself).

Errors
```
12035 Theory ?0 is not present in the current hierarchy
```

SML
```
val get_theory_names : unit -> string list;
val theory_names : unit -> string list;
```

Description These return the list of undeleted theories in the current hierarchy, whether in scope or not. *theory_names* is an alias for *get_theory_names*.

SML
```
val get_theory_status : string -> THEORY_STATUS;
```

Description This returns the status of the indicated theory.

Errors
```
12035 Theory ?0 is not present in the current hierarchy
```

SML
```
val get_theory : string -> THEORY;
val get_theory_info : string -> THEORY_INFO;
```

Description These functions return the data structures associated with a theory in the logical kernel.

Errors

12035 Theory ?0 is not present in the current hierarchy

SML
```
val get_thms : string -> (string list * THM) list;
val get_thm_dict : string -> THM OE_DICT;
```

Description *get_thms* returns all the theorems stored in the indicated theory together with the keys under which they are stored.

get_thm_dict returns the mapping of keys to theorems represented as an order-preserving efficient dictionary.

Errors

12035 Theory ?0 is not present in the current hierarchy

SML
```
val get_thm : string -> string -> THM;
```

Description *get_thm theory key* returns the theorem with key *key*, found in theory *theory*.

To improve performance, this function uses a cache containing the values of previous calls. This cache is rebuilt when *open_theory* is called, by removing entries that have gone out of scope. Opening a theory such as *basic_hol* that is low down in the theory hierarchy will reclaim the memory occupied by the cache.

Errors

12035 Theory ?0 is not present in the current hierarchy
12006 Theory ?0 does not have a theorem with key ?1
12010 Theory ?0 is not in scope

SML
```
val get_types : string -> TYPE list;
```

Description This returns (canonical applications of) all the type constructors stored on a theory.

Errors

12035 Theory ?0 is not present in the current hierarchy

SML
```
val get_type_arity : string -> int OPT;
```

Description If a type with the given name is in scope, then its arity is returned, otherwise *Nil*.

Uses This is likely to be often used just as a rapid test for a type being in scope.

See Also *get_type_info*

SML
```
val get_type_keys : string -> E_KEY list;
```

Description This returns the efficient dictionary keys that represent the names of the type constructors stored in a theory.

Errors

12035 Theory ?0 is not present in the current hierarchy

SML
```
val get_type_theory : string -> string;
```

Description *get_type_theory ty* returns the name of the theory in which the type constructor *ty* is defined.

Errors

12202 There is no type constructor called ?0 in the current context

SML
```
val get_user_datum : string -> string -> USER_DATUM;
```

Description *get_user_datum thy key* returns the value stored in the user data slot allocated to *key* in the theory *thy*, if any.

Errors

12035 Theory ?0 is not present in the current hierarchy
12009 No user data stored under key ?0 in theory ?1

SML
```
val is_theory_ancestor : string -> string -> bool;
```

Description *is_theory_ancestor thy1 thy2* returns true if *thy1* is an ancestor of *thy2* within the current hierarchy.

Errors

12035 Theory ?0 is not present in the current hierarchy

This failure arises if either theory name is not present in the current hierarchy.

SML
```
val kernel_interface_diagnostics : bool -> {
        clean_flag : bool,
        const_thys : int list E_DICT list,
        type_thys: int list E_DICT list,
        int_thy_names : int E_DICT,
        in_scope : int list};
```

Description This function can be used to examine and optionally reset internal state used by the kernel interface module. It is intended for diagnostic purposes. If the argument is *false*, it just returns a representation of the state; if *true*, it also sets the internal state so that the next call on any operation such as *get_const_theory* will cause the state to be recalculated.

SML
```
val list_save_thm : (string list * THM) -> THM
```

Description *list_save_thm(keys, thm)* causes *thm* to be save under the keys *keys* in the current theory. The saved theorem is returned as the function's result. If there is a conjecture stored under any of the keys in the current theory, the theorem must prove each such conjecture, i.e., its conclusion must be the same as the conjecture and it must have an empty assumption list.

See Also *new_conjecture*, *is_proved_conjecture*

Errors

6031 Key list may not be empty
6037 Theory ?0 is locked
6039 Key ?0 has already been used for a theorem in theory ?1
6071 Theory ?0 is a read−only ancestor
103101 This theorem does not prove the conjecture stored under key ?0

SML
```
val lock_theory : string -> unit;
```

Description *lock_theory thy* causes *thy* to be *locked.* The contents of a locked theory are protected from further changes. A locked theory may be unlocked using *unlock_theory*(q.v.).

Errors
```
12035  Theory ?0 is not present in the current hierarchy
6037   Theory ?0 is locked
6071   Theory ?0 is a read−only ancestor
```

SML
```
val new_axiom : (string list * TERM) -> THM
```

Description *new_axiom*(*keys*, *tm*) stores the boolean term *tm* an axiom in the current theory as an axiom under keys *keys*.

Errors
```
3031   ?0 is not of type ⌜:BOOL⌝
6031   Key list may not be empty
6037   Theory ?0 is locked
6047   Key ?0 has already been used for an axiom in theory ?1
6071   Theory ?0 is a read−only ancestor
```

SML
```
val new_const : (string * TYPE) -> TERM;
```

Description *new_const* (*name*, *type*) introduces a new constant (with no defining theorem) called *name*, with most general type *type*, into the current theory.

Errors
```
6037   Theory ?0 is locked
6049   There is a constant called ?0 already in scope
6063   There is a constant called ?0 in the descendants of the
       current theory
6071   Theory ?0 is a read−only ancestor
```

SML
```
val new_parent : string -> unit;
```

Description Adds the given parent theory to the list of parents of the current theory, considered as a set. It will fail if the parent theory does not exist; is already a parent of the current theory; or if making it a parent would cause a clash by bringing a new theory into scope (perhaps the new parent itself) that declares a new type or constant that is already in scope, or is declared in the descendants of the current theory.

Errors
```
12035  Theory ?0 is not present in the current hierarchy
6037   Theory ?0 is locked
6067   Making ?0 a parent would cause a clash
6071   Theory ?0 is a read−only ancestor
6082   Theory ?0 is already a parent
6084   Suggested parent ?0 is a child of the current theory
```

SML

```
val new_spec : (string list * int * THM) -> THM;
```

Description *new_spec* (*keylist*, *ndef*, '$\vdash \exists x\_1, \ldots, x\_n \bullet p[x\_1, \ldots, x\_n]$') will introduce *ndef* new constants named and typed from the *x_i*. It will also save a defining theorem under each of the keys in *keylist* in the current theory of the form '$\vdash p[c\_1, \ldots, c\_n]$' where *c_i* is the constant with the name and type of *x_i*. If either the constant or theorem introduction fails then the function will not change the current theory.

Errors

```
6016  Existentially bound variable ?0 is repeated in theorem ?1
6031  Key list may not be empty
6037  Theory ?0 is locked
6044  Must define at least one constant
6049  There is a constant called ?0 already in scope
6051  Key ?0 has already been used for a definition in theory ?1
6053  ?0 must not have assumptions
6056  ?0 is a free variable in ?1
6062  ?0 are free variables in ?1
6060  ?0 is not of the form: '⊢ ∃ x1 ... xn • p[x1,...,xn]'
      where the ⌜xi⌝ are variables, and n( = ?1) is the number of
      constants to be defined
6061  the body of ?0 contains type variables not found in type
      of constants to be defined, the variables being: ?1
6063  There is a constant called ?0 in the descendants of the
      current theory
6071  Theory ?0 is a read−only ancestor
6081  Sets of type variables in ?0 and ?1 differ
```

SML

```
val new_theory : string -> unit;
```

Description *new_theory thy* adds a new, empty, theory called *thy* to the theory database. The empty theory has no declarations within it, but does have the current theory as its sole parent. The new theory then becomes the current theory.

Errors

```
6040  Theory ?0 is already present in current theory hierarchy
```

SML

```
val new_type_defn :
        (string list * string * string list * THM) -> THM;
```

Description *new_type_defn* (*keys*, *name*, *typars*, *defthm*) declares a new type with name *name*, and arity the length of *typars*. It creates a defining theorem for the type, saves it in the current theory under the keys *keys*. It returns the defining theorem. *defthm* must be a valid well-formed theorem of the form:

$\vdash \exists\ x : type \bullet p\ x$

with no assumptions. The defining theorem will then be of the form:

$\vdash \exists\ f : typars\ name \to type \bullet$
$\quad TypeDefn\ (p\text{: } type \to BOOL)\ f$

where *TypeDefn* asserts that its predicate argument p is non-empty, and its function argument f is a bijection between the new type and the subset of *type* delineated by p.

Errors

6031	*Key list may not be empty*
6034	*There is a type called ?0 in the descendants of the current theory*
6037	*Theory ?0 is locked*
6045	*There is a type called ?0 already in scope*
6052	*Key ?0 has already been used for an type definition theorem in theory ?1*
6053	*?0 must not have assumptions*
6054	*?0 is not of the form:* '$\vdash \exists\ x \bullet px$'
6055	*?0 is not of the form:* '$\vdash \exists\ x \bullet p\ y$' *where* ⌜$x$⌝ *is a variable*
6056	*?0 is a free variable in ?1*
6062	*?0 are free variables in ?1*
6057	*?0 contains type variables not found in type variable parameter list,* *type variables being: ?1*
6071	*Theory ?0 is a read−only ancestor*
6079	*?0 repeated in type parameter list*
6080	*?0 is not of the form:* '$\vdash \exists\ x \bullet p\ y$' *where* ⌜$x$⌝ *equals* ⌜$y$⌝

SML

```
val new_type : (string * int) -> TYPE;
```

Description *new_type* (*name*, *arity*) introduces a new type constructor (with no defining theorem) called *name* with arity *arity* into the current theory. The function returns the new type with sufficient arguments $'1, '2, \ldots$ to provide a well-formed type.

Errors

6034	*There is a type called ?0 in the descendants of the current theory*
6037	*Theory ?0 is locked*
6045	*There is a type called ?0 already in scope*
6071	*Theory ?0 is a read−only ancestor*
6088	*The arity of a type must be* ≥ 0

SML

```
val on_kernel_state_change : (KERNEL_STATE_CHANGE -> unit) -> unit
```

Description *on_kernel_state_change f* nominates *f* to be called whenever the theory database is modified by a function from the signature *KernelInterface*. The argument to *f* encodes the operation which caused the modification together with its arguments and certain other additional information. A list of such functions is maintained, and the new function is put at the end of the list, which means it may, if desired undo or overwrite the effects of a function nominated by an earlier call of *on_kernel_state_change*.

Functions handled by *on_kernel_state_change* should not be coded to raise errors that are not handled by themselves, as the handler will not catch such errors either. If the function is to prevent a change from happening *before_kernel_state_change* should be used instead.

See Also *KERNEL_STATE_CHANGE*, *before_kernel_state_change*

SML

```
val open_theory : string -> unit;
```

Description All specification and proof work is carried out in the context of some theory, referred to as the current theory. *open_theory thy* makes an existing theory *thy* the current theory.

Errors

```
6017  Theory ?0 may not be opened (it is not a descendant of ?1 which must be in scope)
12035 Theory ?0 is not present in the current hierarchy
```

Certain theories created when the system is constructed may not be subsequently opened, and attempts to open them give rise to error 6017.

SML

```
val pending_reset_kernel_interface : unit -> unit -> unit;
```

Description This function, applied to () takes a "snapshot" of the current state of the kernel interface module (comprising the"On Kernel State Change", "Before Kernel State Change" and "On Kernel Inference" functions). The resulting snapshot, when applied to () will restore these functions to their state at the time of making the snap shot.

Uses To assist in saving the overall system state.

SML

```
val save_thm : (string * THM) -> THM
```

Description *save_thm(key, thm)* causes *thm* to be save under the key *key* in the current theory. The saved theorem is returned as the function's result. If there is a conjecture stored under the same key in the current theory, the theorem must prove the conjecture, i.e., its conclusion must be the same as the conjecture and it must have an empty assumption list.

See Also *new_conjecture*, *is_proved_conjecture*

Errors

```
6037   Theory ?0 is locked
6039   Key ?0 has already been used for a theorem in theory ?1
6071   Theory ?0 is a read-only ancestor
103101 This theorem does not prove the conjecture stored under key ?0
```

SML
```
val set_user_datum : (string * USER_DATUM) -> unit;
```

Description *set_user_datum(key, ud)* assigns the new value *ud* to the user data slot allocated to *key* in the current theory. If an old value was present it will be overwritten.

Errors
```
6037  Theory ?0 is locked
6071  Theory ?0 is a read-only ancestor
```

SML
```
val simple_new_defn : (string list * string * TERM) -> THM;
```

Description *simple_new_defn (keys, name, value)* declares a new constant with name *name*, and with most general type being the type of *value* in the current theory. It creates an equational theorem (i.e. of the form '⊢*name* = *value*'), and saves it as a definition under keys *keys* in the current theory, provided the theorem is well-formed. If either the constant or theorem introduction fails then the function does not change the current theory. The body of *value* may not contain type variables that are not in the type of *value* itself.

Errors
```
6031  Key list may not be empty
6037  Theory ?0 is locked
6049  There is a constant called ?0 already in scope
6051  Key ?0 has already been used for a definition in theory ?1
6058  the body of ?0 contains type variables not found in type of term itself,
      the variables being: ?1
6059  ?0 contains the following free variables: ?1
6063  There is a constant called ?0 in the descendants of the
      current theory
6071  Theory ?0 is a read-only ancestor
```

SML
```
val string_of_thm : THM -> string;
```

Description This returns a display of a theorem in the form of a string, with no inserted new lines, suitable for use with *diag_string* and *fail*.

See Also *format_thm*, a formatted string display of a theorem.

SML
```
val thm_fail : string -> int -> THM list -> 'a;
```

Description *thm_fail area msg thml* first creates a list of functions from *unit* to *string*, providing displays of the list of theorems. It then calls *fail* with the *area*, *msg* and this list of functions. This allows theorems to be presented in error messages.

SML
```
val thm_theory : THM -> string;
```

Description *thm_theory thm* returns the name of the theory which was current when *thm* was proven. This will succeed even if the theory is out of scope, but not if the theory has been deleted.

Errors
```
12007 ?0 proven in theory with internal name ?1,
      which is not present in current hierarchy
```

SML
```
val unlock_theory : string -> unit;
```

Description *unlock_theory thy* causes the locked theory *thy* to be unlocked, so that the contents of *thy* may be changed.

Errors
```
12035 Theory ?0 is not present in the current hierarchy
6068  Theory ?0 has not been locked
```

SML
```
val valid_thm : THM -> bool;
```

Description This function uses the check for the validity of theorems: returning true if valid and false otherwise: it cannot raise exceptions.

Uses To preempt errors caused by the primitive inference rules, which raise uncatchable errors when given invalid theorems, and so return more helpful error messages.

6.4 Conjectures Database

SML

```
val is_proved_conjecture: string -> string -> bool;
val get_proved_conjectures: string -> string list;
val get_unproved_conjectures: string -> string list;
```

Description *is_proved_conjecture thy key* returns true if the conjecture with key *key* in theory *thy* has been proved (i.e., there is a theorem stored under the same key in the theory which has the conjecture as its conclusion and has no assumptions).

get_proved_conjectures thy (resp. *get_unproved_conjectures thy*) returns the list of conjectures in theory *thy* which have (resp. have not) been proved in the sense described above.

See Also *save_thm*, *list_save_thm*, *new_conjecture*

Errors

```
20601  There is no theory called ?0
103101 This theorem does not prove the conjecture stored under key ?0
103102 The theorem with key ?0 does not prove this conjecture
103103 Theory ?0 is not in scope
103802 There is no conjecture called ?0 in theory ?1
103803 The conjectures database in theory ?0 is corrupt
       (use delete_all_conjectures to clear).
```

SML

```
val new_conjecture : (string list * TERM) -> unit;
val get_conjecture: string -> string -> TERM;
val get_conjectures: string -> (string list * (int * TERM)) list;
val delete_conjecture: string -> TERM;
val delete_all_conjectures: unit -> unit;
```

Description *new_conjecture*(*keys*, *tm*) stores the boolean term *tm* as a conjecture in the current theory under keys *keys*. If any of the keys is also the key of a theorem saved in the current theory, then each such theorem must prove the conjecture, i.e., its conclusion must be the same as *tm* and it must have an empty assumption list.

delete_conjecture key deletes the conjecture stored in the current theory under key *key*. It returns the deleted conjecture.

delete_all_conjectures() deletes all the conjectures stored in the current theory. This may be used if, for some reason, the data structure used to store the conjectures becomes corrupted.

Note, when a constants or a type is deleted from a theory, conjectures that contain the deleted constant or type are automatically deleted from the current theory. Message 103804 is used as a comment to inform the user when this happens.

See Also *save_thm*, *list_save_thm*, *is_proved_conjecture*

Errors

3031	*?0 is not of type ⌜:BOOL⌝*
6031	*Key list may not be empty*
20601	*There is no theory called ?0*
103101	*The theorem ?0 does not prove the conjecture with key ?1*
103801	*Key ?0 has already been used for a conjecture in the current theory*
103802	*There is no conjecture called ?0 in theory ?1*
103803	*The conjectures database in theory ?0 is corrupt (use delete_all_conjectures to clear).*
103804	*Deletion of ?0 has caused deletion of conjecture?1: ?2*

6.5 Theorem Finder

SML
```
datatype 'a TEST =
        TFun of 'a -> bool
|       TAll of 'a TEST list
|       TAny of 'a TEST list
|       TNone of 'a TEST list;

type THM_INFO_TEST = THM_INFO TEST;
```

Description The type *THM_INFO_TEST* is used for the parameters of general theorem finder functions, *gen_find_thm* and *gen_find_thm_in_theories* that represent search criteria. The constructor *TFun* is used to represent a basic criterion. *TAll*, *TAny* and *TNone* construct new criteria from old by conjunction, disjunction and negated disjunction respectively.

See Also *any_substring_tt* etc. (for ways of constructing basic criteria).

SML
```
datatype THM_TYPE = TTAxiom | TTDefn | TTSaved;

type THM_INFO = {
        theory      : string,
        names       : string list,
        thm_type    : THM_TYPE,
        thm         : THM};
```

Description The types *THM_TYPE* and *THM_INFO* are used by the theorem finder functions, *find_thm* etc., to represent information about a theorem stored in a theory. The representation gives: the name of the theory; the name or names under which the theorem is stored; an indicator of whether the theorem is an axiom, a definition or a theorem that has been proved and saved; and the actual theorem.

SML
```
val find_thm : TERM list -> THM_INFO list;
```

Description This is a simple interface for finding theorems. *find_thm pats* searches for any theorems in the current theory and its ancestors that contains subterms matching each of the pattern terms *tms*.

The return value is a list of records containing the conclusion of the theorem and other useful information, see the description of the type *THM_INFO* for more details.

For example, if the theory ℝ of real numbers is in scope, the following will find all theorems containing both real number addition and real number multiplication.

```
find_thm [⌜x + y : ℝ⌝, ⌜x * y : ℝ⌝];
```

See Also *gen_find_thm*

SML

```
val gen_find_thm_in_theories : THM_INFO_TEST -> string list -> THM_INFO list;
val gen_find_thm : THM_INFO_TEST -> THM_INFO list;

val any_substring_tt : string list -> THM_INFO TEST;
val all_substring_tt : string list -> THM_INFO TEST;
val no_substring_tt : string list -> THM_INFO TEST;
val any_subterm_tt : TERM list -> THM_INFO TEST;
val all_subterm_tt : TERM list -> THM_INFO TEST;
val no_subterm_tt : TERM list -> THM_INFO TEST;
val any_submatch_tt : TERM list -> THM_INFO TEST;
val all_submatch_tt : TERM list -> THM_INFO TEST
val no_submatch_tt : TERM list -> THM_INFO TEST
```

Description *gen_find_thm_in_theories* is the general theorem finder function. Its first parameter specifies the search criteria and its second parameter specifies the names of the theories to be searched. It returns a list representing the theorems satisfying the criteria. See the definitions of the parameter and return data types for more details.

gen_find_thm calls *gen_find_thm_in_theories* with the specified search criteria and the list of all ancestors of the current theory as the list of theories to search (this include the current theory). Thus it finds all the theorems that are currently in scope that match the specified criteria.

The remaining functions give convenient ways of specifying typical search criteria. These functions support three kinds of basic criterion: substring search criteria test for a specified string appearing as a substring of the name of the theorem; subterm search criteria test for the presence (up to α-equivalence) in the conclusion of the theorem of a specified subterm; submatch search criteria test for the presence in the conclusition of the theorem of a subterm that is an instance of a specified pattern term. Given a list of strings or terms giving basic criteria, the functions test for theorems satisfying all of the criteria (*all_...*), at least one of the criteria (*any_...*) or none of the criteria (*no_...*). The constructors of the data type *THM_INFO_TEST*, q.v., allow more complex logical combinations of criteria to be built up from these.

For example, the following will find all theorems in scope that have names containing "plus" or "minus" as a substring and that have a conclusion that does not contain any natural number additions.

```
gen_find_thm(TAll[any_substring_tt["plus", "minus"], no_subterm_tt[⌜$+:ℕ → ℕ → ℕ⌝]]);
```

See Also *find_thm*

SPECIFICATION IN HOL

7.1 Constant Specification

SML
```
signature ConstantSpecification = sig
```

Description This is the signature of a structure supporting specification of constants.

SML
```
val consistent_def: THM;
val const_spec_def:THM;
val const_spec_thm: THM;
```

Description These theorems are ML variables bound to the definition of *Consistent*, the definition of *ConstSpec* and the theorem saved with key *const_spec_thm* in theory "basic_hol".

```
val consistent_def = ⊢ ∀ p • Consistent p ⇔ ∃ x • p x
val const_spec_def = ⊢ ∀ p c• ConstSpec p c ⇔ (Consistent p ⇒ p c)
val const_spec_thm = ⊢ ∀ p • ∃ x : 'a • ConstSpec p x
```

SML

```
val const_spec : string list * TERM list * TERM -> THM;
```

Description *const_spec* is used to introduce new constants that satisfy some predicate. *const_spec* (*keys*, *varstructs*, *predicate*) declares constants matching the names and types of those in *varstructs*, with the property that, if *predicate* holds for some list of values, then it holds for the constants in question, with the definition saved under each of *keys*. In addition, if the existence prover held in the current proof context, and accessed by *current_cs_∃_conv*, can prove the predicate is satisfied for some witness (i.e. ⇔ *T*), the specification will instead be made without the caveat. In more detail, if the list of free variables in *varstructs* is *c_1 ... c_n*, and the predicate is of the form *P*[*c_1 ... c_n*] : *BOOL*, then the specification of constants *c_1 ... c_n* (with same names and types as the variables) will be:

```
⊢ ConstSpec (λ (x_1, ..., x_n) • P[x1,...,xn]) (c1,...,cn)
```

where each *x_i* is a variant (see *list_variant*) of *c_i*, not present in the original list of variables, or the predicate, and the *c_i* become constants. However, if we can automatically prove: ∃ *c1 ... cn* • *P*[*c1*,...,*cn*] then instead the specification will be: ⊢ *P* [*c1*,...,*cn*]. There are a number of caveats on the production of the specification: there must be no duplicates in *varstructs*; all the constant's types must have the same set of type variables; there must be no type variables in the body of *predicate* other than those already in the constant's types; and there must be at least one key.

See Also *HOL_axiomatic_recogniser*, *get_spec*.

Errors

```
3031   ?0 is not of type ⌜:BOOL⌝
6031   Key list may not be empty
6037   Theory ?0 is locked
6044   Must define at least one constant
6049   There is a constant called ?0 already in scope
6051   Key ?0 has already been used for a definition in theory ?1
6056   ?0 is a free variable in ?1
6062   ?0 are free variables in ?1
6061   the body of ?0 contains type variables not found in type
       of constants to be defined, the variables being: ?1
6063   There is a constant called ?0 in the descendants of the current theory
6071   Theory ?0 is a read−only ancestor
6081   Sets of type variables in ?0 and ?1 differ
46003  Variable name ?0 is repeated in varstructs
46004  ?0 is not a varstruct
```

SML

```
val get_spec : TERM -> THM;
```

Description *get_spec* ⌜*const*⌝ will find the (first) definition or axiom in scope stored under key "name of *const*", in the theory in which the in-scope constant named *const* was defined. A definition will be taken in preference to an axiom in the same theory. It it cannot find such a theorem it will determine the list of languages of the constant (e.g. "HOL" and "Z"). It will then attempt to strip a language name plus a prime, in a case-insensitive manner, from the constant name. If there is one such stripped name it will then search in the same theory for a definition or axiom using this stripped name as a key. If there is no such name it will fail with message 46005, or if more than one such stripped name with matching theorem (a very rare circumstance), it will fail with message 46013. If there is no such constant in scope then the function fails.

get_spec ⌜*const t1 t2 ...*⌝ (i.e. a constant applied to an arbitrary number of arguments) will act as *get_spec* ⌜*const*⌝. If ICL conventions have been followed, then a definition (or axiom) found as above should be the definition of the constant named *const*. In addition, there can only be one definition of a particular constant in scope (though the conventional key might be used elsewhere, or not at all). Further, under such conventions, two constants of different languages but with the same name should not be saved in the same theory.

If the definitional theorem is not of the form:

$\vdash$ *ConstSpec p c*

then the theorem is returned un-processed - if properly saved then it either comes from (*simple_*) *new_defn* , *new_spec*, or predicate *p* was automatically proven consistent by *const_spec*.

Otherwise, the function will seek for a theorem or axiom stored with key *const* ^ "*_consistent*", starting at the theory in which the definition was found, and working "out" to the current theory. If conventions have been followed this theorem should be of the form:

$\Gamma \vdash$ *Consistent p*

(Ideally there should be no assumptions in the theorem, but the function caters for their presence.) If a theorem of this form is found then the function returns:

β_rule ‘$\Gamma \vdash$ *p c*‘

If not, then the function returns a theorem of the form:

β_rule ‘*Consistent p* $\vdash$ *p c*‘

No language-based key name stripping will be attempted for consistency theorem names.

See Also *push_consistency_goal* to set the appropriate consistency goal, (Γ, ⌜*Consistent p*⌝), for a given constant name.

Errors

46005	There is no constant with name ?0 in scope
46006	There is no definition or axiom with key ?0 in the declaration theory of the constant
46009	?0 is not a constant, or a constant applied to some arguments
46013	?0 has more than one possible definition: ?1

SML

```
val HOL_const_recogniser : string * string * Lex.INPUT list * string
        -> THM;
```

Description From the argument (which will be supplied by the reader/writer) this function will derive a list of varstructs, and a predicate. These will then be passed to *const_spec* (q.v.) which will attempt the requested specification. The keys under which the result will be saved are just the names of the variables concerned.

Uses As an interface between the HOL reader/writer and *const_spec* - it is not intended for direct use. It is used to process text of the form

Example Constant Specification

```
        decls
----------------------------------------
        pred
```

Errors

```
46000  Input not of right form
```

Errors from *const_spec* may also occur: the messages area of origin will not be changed.

SML

```
val push_consistency_goal: TERM -> unit;
```

Description *push_consistency_goal* ⌜*const*⌝ will first determine the specification theorem of *const*, by executing *get_spec*. The *const* may either be a constant, or a constant applied to a list of arguments. If this theorem has an assumption, it will then push that specification assumption onto the stack of subgoals (using *push_subgoal*, q.v.), as a goal with no assumptions. By how *get_spec* is designed, this (single) assumption will be of the form

```
⌜Consistent (λ vs[x1,...,xn]•p[x1,...,xn])⌝
```

or the consistency has already been proven, and saved, under some assumptions. Only in the former case will the function continue: it will apply a tactic (that may be undone by *undo*) which rewrites the goal to precisely:

```
([], ⌜∃ vs[x1,...,xn]•p[x1,...,xn]⌝)
```

If not, the function fails.

See Also *save_consistency_thm* to save the result in a conventional manner.

Errors

```
46005  There is no constant with name ?0 in scope
46006  There is no definition or axiom with key ?0 in
       the declaration theory of the constant
46007  Specification of ?0 is not of the form: ‘Consistent (λ vs[x1,...,xn]•p[x1,...,xn]) ⊢ ...‘
46009  ?0 is not a constant, or a constant applied to some arguments
```

SML

```
val save_consistency_thm : TERM -> THM -> THM;
```

Description *save_consistency_thm const thm* expects *thm* to be of the form

$\Gamma \vdash Consistent\ (\lambda\ vs[x1,...,xn] \bullet p[x1,...,xn])$

where *vs* forms a varstruct from the simple variables x_i. The predicate proven consistent is then expected to be the same as predicate in the *const_spec* definition of *const*. If so, the theorem will be saved under the keys *c1_consistent*, ..., *cn_consistent*, where the *c_i* are the names of all the constants defined in the definition in question. The *const* may either be a constant, or a constant applied to a list of arguments.

Ideally there should be no assumptions in the theorem, but the function caters for their presence.

See Also *push_consistency_goal* to set a goal suitable for saving with this theorem.

Errors

6037	Theory ?0 is locked
6039	Key ?0 has already been used for a theorem in theory ?1
6071	Theory ?0 is a read−only ancestor
46005	There is no constant with name ?0 in scope
46006	There is no definition or axiom with key ?0 in the declaration theory of the constant
46008	?0 is not of the form: ⌜Consistent (λ vs[x1,...,xn]•p[x1,...,xn])⌝
46009	?0 is not a constant, or a constant applied to some arguments
46011	?0 does not match definition of ?1
46012	?0 is not of the form: ⌜ConstSpec p c⌝

If even one key causes a failure, the theorem will not be saved at all (the behaviour of *list_save_thm*).

7.2 Automatic Existence Proofs

SML
signature **ExistenceProofs** = *sig*

Description This is the signature of a structure supporting automatic existence proofs.

SML
val **basic_prove_∃_conv** : *CONV*;

Description This conversion uses its internal mechanisms, and material stored in the current proof context, to attempt to prove a "simpler" term equal to its argument, which must be of the form ⌜∃...⌝. It will simplify by:

- Converting paired existential quantifiers into simple ones (see *all_∃_uncurry_conv*).
- Removing existential and universal quantifiers not used in the body of their quantification.
- Distributing existential quantifiers over ∧ and ∨ as far as it is able (see *simple_∃_∧_conv* and *simple_∃_∨_conv*).
- Eliminating an existential quantifier that is equated to a term within the body of the quantification. Implicit equations (e.g. an implicit ⇔ *T* or by ¬) are also handled (see *simple_∃_equation_conv* for caveats).
- "Pulling out" universal quantifiers through existential quantifiers if the existential is a function applied in all its instances to the universally quantified variables (see *simple_∃_∀_conv1*), or variable structures formed by data constructors accessed by *current_ad_∃_vs_thms* in the current proof context. Redistribution of universals and data constructions may be done to allow this simplification to apply (see *∀_∧_conv*).
- Uncurrying existential quantifiers that are curried functions.
- Simplifying existential quantifiers that match the pattern of use of recursive functions held by *current_ad_∃_cd_thms* in the current proof context.
- Traversing subterms reached from the outside of the term through (perhaps paired) existential and universal quantifiers, and ∧ or ∨ operators.

It will repeat its simplification attempts until it can go no further.

Conversion

$$\frac{}{\vdash (\exists\ decls \bullet\ pred) \Leftrightarrow simpler} \quad \begin{array}{l} basic\_prove\_\exists\_conv \\ \ulcorner \exists\ decls \bullet\ pred \urcorner \end{array}$$

Uses To implement *basic_prove_∃_rule* and *basic_prove_∃_tac* (q.v), as an appropriate argument for *set_cs_∃_conv*, and as a stand-alone tool.

Errors
58001 ?0 is not of the form: ⌜∃ *decls*• *pred*⌝
58002 Failed to simplify ?0

SML

val **basic_prove_∃_rule** : $TERM$ −> THM;

Description This will attempt to simplify its argument to T, by using *basic_prove_∃_conv* (q.v.). If it succeeds, it returns the theorem with $\Leftrightarrow T$ stripped off.

Rule

$$\frac{}{\vdash \exists\ decls \bullet\ pred} \quad basic\_prove\_\exists\_rule\ \ulcorner \exists\ decls \bullet\ pred \urcorner$$

See Also *basic_prove_∃_conv*, *basic_prove_∃_tac*

Errors

58001 ?0 is not of the form: $\ulcorner \exists\ decls \bullet\ pred \urcorner$
58002 Failed to simplify ?0
58003 Failed to prove ?0

We distinguish between no simplification, and failing to prove a term, to indicate whether *basic_prove_∃_conv* may be of initial use in a manual proof of the term.

SML

val **basic_prove_∃_tac** : $TACTIC$;

Description This will attempt to prove its goal, by using *basic_prove_∃_conv* (q.v.). If *basic_prove_∃_conv* fails, or only partially succeeds, then the tactic will fail.

Tactic

$$\frac{\{\ \Gamma\ \}\ \exists\ decls \bullet\ pred}{} \quad basic\_prove\_\exists\_tac$$

See Also *basic_prove_∃_conv*, *basic_prove_∃_rule*

Errors

58004 Goal is not of the form: $\{\ \Gamma\ \}\ \exists\ decls \bullet\ pred$
58005 Failed to simplify goal
58006 Failed to prove goal

We distinguish between no simplification, and failing to prove a term, to indicate whether *basic_prove_∃_conv* may be of initial use in a manual proof of the term.

SML

```
val evaluate_∃_cd_thm : THM ->
        (TERM list * int list * TYPE * (TERM list)list * THM);
```

Description This gives the sophisticated user direct access to the standard internal mechanism set by *pp′set_eval_ad_∃_cd_thm*. Its input theorem is of the "standard" form of a primitive recursion theorem or pattern matching theorem:

```
∀ fn1 fn2 ...• ∃ f•
(∀ a11 ...• f (dc1[a11,...]) = fn1 (patt11[f,a11...]) (patt12[f,a11,...]) ...) ∧
(∀ a21 ...• f (dc2[a21,...]) = fn1 (patt21[f,a21...]) (patt22[f,a21,...]) ...) ∧
...
```

Compare this with:

```
prim_rec_thm = ⊢ ∀ z s• ∃1 f• (f 0 = z) ∧ (∀ n• f (n + 1) = s (f n) n)
```

The *fn_i* (in the example: *z* and *s*) are the new functions which express the behaviour of each data constructor. *f* (*f*) is the function defined by clausal definitions, note that unique existence is not necessary. The *a_ij* (*n* in the second conjunct) are the free variables of the data construction. The *dc_i* (*n + 1* and *0*) are data constructions, as in the example given these do not need to be the original data constructors of a type, just applications of constants and variables. The *patt_ij* (*f n*, *n*) are the arguments to the new functions. They may involve uses of the clausally defined function.

The result of this function is: the list of data constructions, a list of free variable counts in each data construction, the type of *f*, a list of free variables in each data construction, and a theorem of the form:

```
⊢ ∀ pred1 ...•
(∃ f• (∀ a11 ...• pred1 (dc1[...]) (patt1[...]) ..) ∧ ...) ⇔
((∀ x11 ... x1n• ∃ y1• pred1 y11 x11 ...) ∧ ...) ∨
(pp′TS(∃ f• (∀ a11 ...• pred1 (dc1[...]) (patt1[...]) ..) ∧ ...))
```

This is in a convenient form for proving the existence of clausally defined functions, in particular recursive functions within *basic_prove_∃_conv*. The *pred_i*, *x_ij* and *y_i* are generated by *gen_vars*. *pp′TS* is an identity function, with defining theorem in theory "misc" of key "pp'ts_def".

Errors

58007	*?0 is not of the form*: '⊢ ∀ ...• ∃ f• ...' *or* '⊢ ∀ ...• ∃1 f• ...'
58008	*?0 has a conjunct not of the form*: ⌜∀ ...• f dc = ...⌝
58009	*?0 has a conjunct not of the form*: ⌜∀ ...• f dc = ...⌝ *where f is the function whose existence is to be proven*
58021	*Failed to prove new theorem based on ?0*
58023	*?0 has a conjunct not of the form*: ⌜∀ ...• f dc = P ...⌝
58024	*?0 has the form*: '⊢ ∀ ... fni ...• B' *where fni does not appear in any of the expected positions in B*

SML

val **simple_∃_equation_conv** : $CONV$;

Description This conversion eliminates a simple existential quantifier that is equated to a term within the body of the quantification. The term equated with must not contain the existential quantifier, nor may it have free variables that are bound in the body of the existential quantifier, other than in an outer existential. The equation must also be "reached" only through existential and universal quantifications, ∧. Implicit equations (e.g. x being an implicit $x \Leftrightarrow T$ or by $\neg\ y$ giving an implicit $y \Leftrightarrow F$) are also handled.

Conversion

$$\frac{}{\vdash (\exists\ f\bullet\ P\ [(f\ =\ tm),\ f]) \Leftrightarrow P\ [T,\ tm]} \quad \begin{array}{l} simple\_\exists\_equation\_conv \\ \ulcorner\exists\ f\bullet\ P\ (f\ =\ tm)\ f\ \urcorner\end{array}$$

Some simplification will also take place if any of the following apply where T replaced the selected equation:

$[\wedge\_rewrite\_thm,\ \exists\_rewrite\_thm,\ \forall\_rewrite\_thm,$
$eq\_rewrite\_thm,\ \Rightarrow\_rewrite\_thm,\ \neg\_rewrite\_thm]$

Example

$simple\_\exists\_equation\_conv\ \ulcorner\exists\ f\bullet\ f\ =\ x\urcorner = \vdash (\exists\ f\bullet\ f\ =\ x) \Leftrightarrow\ T$

$simple\_\exists\_equation\_conv\ \ulcorner\exists\ f\bullet\ x\ =\ f\urcorner = \vdash (\exists\ f\bullet\ x\ =\ f) \Leftrightarrow\ T$

$simple\_\exists\_equation\_conv\ \ulcorner\exists\ f\bullet\ (f\ =\ x)\ \wedge\ P\ f\urcorner =$
$\quad \vdash (\exists\ f\bullet\ (f\ =\ x)\ \wedge\ P\ f) \Leftrightarrow P\ x$

$simple\_\exists\_equation\_conv\ \ulcorner\exists\ f\ g\bullet\ (f\ =\ g)\ \wedge\ P\ f\urcorner =$
$\quad \vdash (\exists\ f\ g\bullet\ (f\ =\ g)\ \wedge\ P\ f) \Leftrightarrow \exists\ g\bullet P\ g$

$simple\_\exists\_equation\_conv\ \ulcorner\exists\ f\bullet\ f\ \wedge\ P\ f\urcorner =$
$\quad \vdash (\exists\ f\bullet\ f\ \wedge\ P\ f) \Leftrightarrow P\ T$

$simple\_\exists\_equation\_conv\ \ulcorner\exists\ f\bullet\ \neg\ f\ \wedge\ P\ f\urcorner =$
$\quad \vdash (\exists\ f\bullet\ \neg\ f\ \wedge\ P\ f) \Leftrightarrow P\ F$

See Also *basic_prove_∃_conv* for a more powerful existence prover.

Errors

3034 ?0 is not of form: ⌜∃ var • body⌝
58013 Cannot find an equation in ?0 to simplify
58022 Could not prove ?0 by selected witness ?1

SML

```
val simple_∃_∧_conv: CONV;
```

Description This conversion will push simple existential quantifiers into a term body that consists of nested conjunctions when not all the conjuncts have all of the quantified variables free in them. It will prove the input equal to a term formed by conjoining the list of existentially quantified groupings of conjuncts, grouped by possession of a quantified variable free in each of the group. It may not necessarily form the "most pushed in" form with each conjunct minimally bound, but will do some pushing in if any is available.

Conversion

$$\frac{}{(\exists\ x1\ ...\ xn\bullet\ ....\ \wedge\ P[y1...ym]\ \wedge\ ....)\ \Leftrightarrow\ \langle pushed\ in\ form\rangle}\qquad simple\_\exists\_\wedge\_conv\ \ulcorner\exists\ x1\ ...\ xn\bullet\ ...\wedge\ P[y1...ym]\ \wedge...\urcorner$$

where the y_i are a subset of the x_j.

Example

```
simple_∃_∧_conv ⌜∃ x• P[x] ∧ Q⌝ =
        ⊢ (∃ x• P[x] ∧ Q) ⇔ (∃ x• P[x]) ∧ Q

simple_∃_∧_conv ⌜∃ x• P ∧ Q[x]⌝ =
        ⊢ (∃ x• P ∧ Q[x]) ⇔ (∃ x• Q[x]) ∧ P

simple_∃_∧_conv ⌜∃ x y• P[x] ∧ Q[y]⌝ =
        ⊢ (∃ x y• P[x] ∧ Q[y]) ⇔ (∃ y• Q[y]) ∧ (∃ x•P[x])

simple_∃_∧_conv ⌜∃ x y z• P[x, z] ∧ Q[y, z]⌝ =
        ⊢ (∃ x y z• P[x, z] ∧ Q[y, z]) ⇔
        ∃ z• (∃ y• Q[y, z]) ∧ (∃ x•P[x, z])
```

The original conjunctive structure and ordering is lost.

See Also *∃_uncurry_conv* to uncurry any paired ∃-abstractions. *simple_∃_∨_conv*.

Errors

```
58012  ?0 is not of the form: ⌜∃ x1 ...• a ∧ b⌝
58014  All conjuncts of ?0 have all existentially quantified variables
       free within them
```

SML

val **simple_∃_∨_conv**: $CONV$;

Description This conversion will push simple existential quantifiers into a term body that consists of nested disjunctions. It will prove the input equal to a term formed by disjoining the list of disjuncts, existentially quantified by any of the original quantifiers that are free in that disjunct.

Conversion

$$\frac{}{\vdash (\exists\ x1\ ...\ xn\bullet\ P[y1 ... ym] \lor Q[z1 ... zo]) \Leftrightarrow (\exists\ y1\ ...\ ym\bullet\ P\ [y1\ ...\ ym]) \lor (\exists\ z1\ ...\ zo\bullet\ P\ [z1\ ...\ zo])} \quad simple\_\exists\_\lor\_conv \ \ulcorner\exists\ x1\ ...\ xn\bullet\ P\ [y1\ ...\ ym] \lor Q\ [z1\ ...zo]\urcorner$$

where $\{y1 ... ym\} \subseteq \{x1 ... xn\}$, and $\{z1 ... zo\} \subseteq \{x1 ... xn\}$. The conversion will actually distribute over an arbitrary disjunctive structure, though the result is "flattened".

Example

$simple\_\exists\_\lor\_conv\ \ulcorner\exists\ x\ y\bullet\ f\ x \lor g\ y\urcorner =$
$\quad\vdash (\exists\ x\ y\bullet\ f\ x \lor g\ y) \Leftrightarrow (\exists\ x\bullet\ f\ x) \lor (\exists\ y\bullet\ g\ y)$

$simple\_\exists\_\lor\_conv\ \ulcorner\exists\ x\ y\bullet\ f\ x \lor g\ y \lor h\ x\ y \lor i\urcorner =$
$\quad\vdash (\exists\ x\ y\bullet\ f\ x \lor g\ y \lor h\ x\ y \lor i) \Leftrightarrow$
$\qquad(\exists\ x\bullet\ f\ x)$
$\qquad\lor (\exists\ y\bullet\ g\ y)$
$\qquad\lor (\exists\ x\ y\bullet\ h\ x\ y)$
$\qquad\lor\ i$

See Also $\exists\_uncurry\_conv$ to uncurry any paired ∃-abstractions. $simple\_\exists\_\land\_conv$.

Errors

58010 ?0 is not of the form: $\ulcorner\exists\ x1\ ...\bullet\ a \lor b\urcorner$
58015 Unexpected feature of ?0 caused failure

SML

val ∀_∧_**conv**: $CONV$;

Description This conversion will push universal quantifiers into a term body that consists of nested conjunctions. It will prove the input equal to a term formed by conjoining the list of conjuncts, universally quantified by any of the original quantifiers that are free in that conjunct.

Conversion

$$\frac{}{\begin{array}{c}\vdash (\forall\ x1\ ...\ xn\bullet\ P[y1...\ ym] \land Q[z1...zo]) \\ \Leftrightarrow \\ (\forall\ y1\ ...\ ym\bullet\ P\ [y1...ym]) \land \\ (\forall\ z1\ ...\ zo\bullet\ P\ [z1...zo])\end{array}} \quad \begin{array}{l}\forall\_\land\_conv \\ \ulcorner\forall\ x1\ ...\ xn\bullet \\ P\ [y1\ ...\ ym] \\ \land\ Q\ [z1\ ...\ zo]\urcorner\end{array}$$

where $\{y1...ym\} \subseteq \{x1...xn\}$, and $\{z1...zo\} \subseteq \{x1...xn\}$. The conversion will actually distribute over an arbitrary conjunctive structure, though the result is "flattened". The conversion will work with paired universal quantifiers, and will still return a result even if it only acts as *all_∀-_uncurry_conv* (q.v.), and does not redistribute the quantifiers any further.

Example

```
∀_∧_conv ⌜∀ x y• f x ∧ g y⌝ =
        ⊢ (∀ x y• f x ∧ g y) ⇔ (∀ x• f x) ∧ (∀ y• g y)

∀_∧_conv ⌜∀ x y• f x ∧ g y ∧ h x y ∧ i⌝ =
        ⊢ (∀ x y• f x ∧ g y ∧ h x y ∧ i) ⇔
                (∀ x• f x)
                ∧ (∀ y• g y)
                ∧ (∀ x y• h x y)
                ∧ i
```

See Also *∀_uncurry_conv* to uncurry any paired ∃-abstractions.

Errors

58020 ?0 is not of the form: ⌜∀ x1 ...• a ∧ b⌝

7.3 Product Types

SML

```
signature ProductTypes = sig
```

Description This is the signature of a structure supporting the introduction of HOL product types.

SML

```
val cached_labelled_product_rule : int -> THM;
```

Description *cached_labelled_product_rule n* proves a theorem about an n-tuple, where *n* is some positive integer. This states that if some type is isomorphic to an n-tuple (an iterated binary product) then a constructor, and projection functions exist for this type. The theorem will be stored in the current cache theory (see *get_cache_theories*) under the key "nCachedLabelledProduct", unless *n = 1*, when it is "built-it". However, if the theorem desired is already in a cache theory in scope, then it just returns the stored existence theorem.

For storing the theorem current cache theory will be found by *force_get_cache_theory*. If no writable cache theory is in scope this will attempt to use *new_parent* to bring one in scope, and failing that will declare the current theory to be a cache theory.

Example

```
> cached_labelled_product_rule 2;

⊢ (∃ abs rep • (∀ a • abs(rep a) = a) ∧ (∀ r • rep(abs r) = r))
  ⇒
  ∃ 2Tuple •
  ∃ Lab2_1 Lab2_2 •
        ∀ t • ∀ (x1:'1) (x2:'2)•
        (Lab2_1(2Tuple x1 x2) = x1 ∧
        Lab2_2(2Tuple x1 x2) = x2) ∧
        (2Tuple(Lab2_1 t) (Lab2_2 t) = t)
```

See Also *cached_unlabelled_product_rule*

Errors

```
59001  Argument ?0 should have been positive
59002  There is no in scope, writable, cache theory (current theory is ?0)
59003  Failed to prove ?0th labelled cached theorem
59008  Unable to bring a writable cache theory into scope
       or create a new cache theory (current theory is ?0)
59017  Current theory (?0) does not have theory basic_hol as an ancestor
```

Also as the appropriate failures of *save_thm*. Failure 59003 suggests a corrupted cached theorem.

SML

val **cached_unlabelled_product_rule** : *int −> THM*;

Description *cached_unlabelled_product_rule n* proves a theorem about an n-tuple, where *n* is some positive integer. This states that if some type is isomorphic to an n-tuple (an iterated binary product) then a constructor with appropriate properties exists for this type. The theorem will be stored in the current cache theory (see *get_cache_theories*) under the key "nCachedUnlabelled-Product", unless *n = 1*, when it is "built-it". . However, if the theorem desired is already in a cache theory in scope, then it just returns the stored existence theorem.

For storing the theorem the current cache theory will be found by *force_get_cache_theory*. If no writable cache theory is in scope this will attempt to use *new_parent* to bring one in scope, and failing that will declare the current theory to be a cache theory.

Example

```
> cached_unlabelled_product_rule 2;

⊢  (∃ abs rep • (∀ a • abs(rep a) = a) ∧ (∀ r • rep(abs r) = r))
   ⇒
   ∃ 2Tuple •
        (∀ x1 x2• ∀ y1 y2•
        (2Tuple x1 x2 = 2Tuple y1 y2) ⇔ (x1 = y1) ∧ (x2 = y2))
        ∧
        (∀ t• ∃ x1 x2• t = 2Tuple x1 x2)
```

Errors

59001	*Argument ?0 should have been positive*
59002	*There is no in scope, writable, cache theory (current theory is ?0)*
59004	*Failed to prove ?0th unlabelled cached theorem*
59008	*Unable to bring a writable cache theory into scope or create a new cache theory (current theory is ?0)*
59017	*Current theory (?0) does not have theory basic_hol as an ancestor*

See Also *cached_labelled_product_rule* Also as the appropriate failures of *save_thm*. Failure 59004 suggests a corrupted cached theorem, but only for certain indicates an otherwise uncaught error.

SML

val **HOL_lab_prod_recogniser** : *string* ∗ *string* ∗ *string* ∗ *Lex.INPUT list* ∗ *string*
    −> *THM*;

Description This function is called by the reader writer and should not be called by a user directly. The first and second arguments are the start symbol for the recogniser and the language name. The third argument is the name of the type, which appears as the argument. to ⓈHOLLABPROD. The fourth argument is the HOL term represented in a form understood by the lexical analyser. The last argument is the terminating symbol, and should always be a "■". The name of the type and the parsed input is passed to *labelled_product_spec* (q.v.) which will attempt the requested specification. The keys under which the result will be saved are just the names of the variables concerned.

This may also save a theorem concerning tuples to the current cache theory (see *get_cache_theory*).

Uses As an interface between the HOL reader/writer and *labelled_product_spec* - it is not intended for direct use. It is used to process text of the form, e.g.,

Example Labelled Product Specification

Ⓢ*HOLLABPROD TYNAME*
    decl decl;
    decl decl; *decl*
■

Errors

59000 Input not of right form

Errors from *labelled_product_spec* may also occur: the messages area of origin will not be changed.

SML

```
val labelled_product_spec:{tyname:string, tykey:string, conname:string,
        constkeys:string list, labels:(string*TYPE)list, tyvars:(TYPE list)OPT} -> THM;
```

Description This function introduces a labelled product type, and its constructor and projection functions. Its argument's fields are:

tyname The name of the new type.

tykey The key under which the defining theorem for the type will be saved.

conname The name of the single constructor of the new type. It will be a curried function, from each of the label types, in turn, to the new type.

constkeys The keys under which the defining theorem for the constructor *conname* and projection functions will be saved.

labels The list of the names and types of the labelled fields in the new type. A projection function will be defined for each label, a single defining theorem for all will be saved under the keys *constkeys*. It is this defining theorem that is the result of the function call.

tyvars If this field is *Nil* then the type variables in the new type will be ordered as their occurrence in *labels*, otherwise as stated by this parameter.

If the appropriate tuple theorem is not already in scope a call of *cached_labelled_product_rule* (*length labels*) will also be made as a side effect.

Example

```
:> labelled_product_spec {tyname="PAIR", tykey="PAIR", conname="Comma",
        constkeys=["Comma","First","Second","pair_ops_def"],
                labels=[("First",⌜:'f⌝),("Second",⌜:'s⌝)], tyvars=Value[⌜:'s⌝,⌜:'f⌝]};
will cause     ⊢ ∃ f : ('s, 'f) PAIR → ('f × 's) • TypeDefn (λ x • T) f
to be saved under the key "PAIR", and the following
                ⊢ ∀ t • ∀ (x1:'f) (x2:'s)•
                First(Comma x1 x2) = x1 ∧
                Second(Comma x1 x2) = x2) ∧
                Comma(First t) (Second t) = t
under keys "Comma", "First", "Second" and "pair_ops_def", and returned as the result
```

Errors

```
59010  Must be at least one label in list
59014  Failed to introduce type and operators for ?0
59015  Some labels are given types containing type variables not found in
       supplied list of type variables, being: ?0
59018  May not use key ?0 for both type and constructor definition
```

Also as the appropriate errors of *cached_labelled_product_rule*, *new_type_defn* and *new_specification*. Error 59014 suggests a corrupted cache theorem.

SML

```
val set_cache_theories : string list -> string list;
val get_cache_theory : unit -> string;
val force_get_cache_theory : unit -> string;
val get_cache_theories : unit -> string list;
val get_valid_cache_theories : unit -> string list;
```

Description These functions provide the interface to the list of cache theories. *set_cache_theories* stores an uninterpreted list of strings, that are intended to be theory names, returning the previous setting.

The current cache theory is the first string of the cache theory list which is a theory name that is in scope and writable (not locked, an ancestor, or below sealing by *pp'seal_hierarchy*). This is the cache theory that should be **written** to as a cache. This is accessed by *get_cache_theory* which will fail if there is no such theory.

force_get_cache_theory will search for the current cache theory as *get_cache_theory*. If that fails, then it will attempt to create one by trying *new_parent* on each of the writable cache theories that are out of scope. If that fails, then it will attempt to create a new cache theory named *cache'N* (where N is a sequence number). If a new theory is created, its parent will be the theory named at the head of the list of cache theories and it will become a parent of the current theory.

get_cache_theories returns the uninterpreted list of strings last set as the cache theories by *set_cache_theories*.

The current cache theories are the names in the cache theory list of in-scope theories (that need not be writable): this function may return the empty list. This list is accessed by *get_valid_cache_theories*

The scope and existence of the cache theories is determined at the point of applying the function names to ().

These functions support a method of working using cache theories, where only a subset of the ancestors of the current theory are searched for particular portions of contents, and these things are only stored in the current cache theory. This will speed the search for these items by such functions as *cached_labelled_product_rule*.

Errors

59002	*There is no in scope, writable, cache theory (current theory is ?0)*
59008	*Unable to bring a writable cache theory into scope or create a new cache theory (current theory is ?0)*
59020	*A new cache theory ?0 has been created*

SML

```
val unlabelled_product_spec : {tyname:string, tykey:string, conname:string,
        conkeys:string list, tyi:TYPE list, tyvars:(TYPE list)OPT} -> THM;
```

Description This function introduces an unlabelled product type Its argument's fields are:

tyname The name of the new type.

tykey The key under which the defining theorem for the type will be saved.

conname The name of the single constructor of the new type. It will be a curried function, from each of the types *tyi*, in turn, to the new type.

conkeys The keys under which the defining theorem for the constructor *conname* will be saved.

tyi The list of the types of the fields in the new type.

tyvars If this field is *Nil* then the type variables in the new type will be ordered as their occurrence in *labels*, otherwise as stated by this parameter.

If the appropriate tuple theorem is not already in scope a call of *cached_unlabelled_product_rule* (*length tyi*) will also be made as a side effect. This is as part of the speeding up of introducing the new type.

Example

```
:> unlabelled_product_spec {tyname="UPAIR", tykey="UPAIR", conname="UComma",
        conkeys=["UComma","ucomma_def"], tyi=[⌜:'f⌝, ⌜:'s⌝], tyvars=Nil};
will cause
⊢ ∃ f : ('f, 's) UPAIR → ('f × 's) • TypeDefn (λ x • T) f
to be saved under the key "UPAIR", and
⊢       (∀ (x1:'f) (x2:'s)• ∀ y1 y2•
        (UComma x1 x2 = UComma y1 y2) ⇔ (x1 = y1) ∧ (x2 = y2))
        ∧ (∀ t • ∃ x1 x2 • t = UComma x1 x2)

to be saved under the keys "UComma" and "ucomma_def", and returned as the result
```

See Also *labelled_product_spec*

Errors

```
59011  There must be at least one type in list
59014  Failed to introduce type and operators for ?0
59016  The supplied types contain type variables not in the
       supplied list of type variables, being: ?0
59018  May not use key ?0 for both type and constructor definition
```

Also as the appropriate errors of *cached_unlabelled_product_rule* and *new_type_defn*. Error 59014 suggests a corrupted cache theory.

PROOF IN HOL

8.1 General Inference Rules

SML
|*signature* **DerivedRules1** = *sig*

Description This provides the derived rules of inference in Release 001 of ICL HOL. Though other rules of inference may be introduced, this document's signature should provide a core set, at least covering the common rules of natural deduction. It subsumes the inference rules of the abstract data type *THM*.

SML
|*signature* **DerivedRules2** = *sig*

Description This provides the further derived rules of inference for ICL HOL. They are primarily concerned with handling paired abstractions.

SML
|*signature* **Rewriting** = *sig*

Description This provides the derived rewriting rule, conversions and tactics for ICL HOL.

SML
|(* **"illformed_rewrite_warning"** *)

Description This flag modifies the behaviour of *REWRITE_MAP_C* and *ONCE_MAP_WARN_C*. When false (its default) it will not warn of illformed rewriting in subterms, with message 26002, though if no other rewriting occurs then error message 26003 will still be used. If true, then the warning will be given if some rewriting is successful, but elsewhere it is illformed.

SML
|*type* **CANON** (* = *THM* −> (*THM list*) *);

Description This is the type abbreviation for a canonicalisation function; such functions are typically used to derive consequences of a theorem meeting some desired criteria. An example is the rewriting canonicalisations which are used to transform theorems into lists of equational theorems for use in the rewriting conversions, rules and tactics.

Combinators are available to assist in the construction of new canonicalisation functions from old.

See Also *THEN_CAN*, *ORELSE_CAN*, *REPEAT_CAN*, *FIRST_CAN*, *EVERY_CAN* as combinators, *fail_can* and *id_can* as building blocks for the combinators.

SML
|*val* **ALL_SIMPLE_∀_C** : *CONV* −> *CONV*;

Description This conversional applies its conversion argument to the body of a repeated simple universal quantification.

Errors As the failure of the conversion argument.

SML

val **all_simple_∀_elim** : THM −> THM;

Description Specialises all the simple universally quantified variables in a theorem:

Rule

$$\frac{\Gamma \vdash \forall\ x1\ ...\ xn \bullet t[x1,\ ...,\ xn]}{\Gamma \vdash t[x1',\ ...,\ xn']} \qquad all\_simple\_\forall\_elim$$

where $x1', \ldots, xn'$ are renamed from $x1, \ldots, xn$ as necessary to avoid clashes with free variables in the assumption list, or duplicated names in the list of specialisations.

SML

val **ALL_SIMPLE_∃_C** : $CONV$ −> $CONV$;

Description This conversional applies its conversion argument to the body of a repeated simple existential quantification.

Errors As the failure of the conversion argument.

SML

val **all_simple_β_conv** : $CONV$;

Description A conversion to eliminate all instances of simple β redexes in a term, regardless of nesting, or even that the β redex was created as the result of an earlier reduction in the conversion's evaluation.

Rule

$$\frac{}{\vdash t = t'} \qquad \begin{array}{l} all\_simple\_\beta\_conv \\ \ulcorner t \urcorner \end{array}$$

t' is t with all simple β redexes reduced.

Uses This uses an optimised term traversal algorithm, superior in speed to the general term traversal algorithms used with conversions, and should be used in preference to them and *β_conv*.

Errors

7020 ?0 contains no β−redexes

SML

val **all_simple_β_rule** : THM −> THM;

Description Eliminate all instances of simple β redexes in a theorem, regardless of nesting, or even that the β redex was created as the result of an earlier reduction in the rule's evaluation.

Rule

$$\frac{\Gamma \vdash t}{\Gamma \vdash t'} \qquad all\_simple\_\beta\_rule$$

t' is t with all β-redexes reduced.

Errors

7020 ?0 contains no β−redexes

SML

val **ALL_∧_C** : $CONV$ −> $CONV$;
val **ALL_∨_C** : $CONV$ −> $CONV$;

Description These respectively apply their conversion argument to:

- All the conjuncts of a structure of conjuncts (including a term that is not a conjunct at all) failing only if the conversion fails for all the conjuncts.
- All the disjuncts of a structure of disjuncts (including a term that is not a disjunct at all) failing only if the conversion fails for all the disjuncts.

The result is simplified at any conjunct or disjunct where at least one branch had a successful application of the conversion and matches the appropriate theorems of:

$\vdash \forall\ t\bullet\ (T \land t \Leftrightarrow t) \land (t \land T \Leftrightarrow t) \land \neg\ (F \land t) \land \neg\ (t \land F) \land (t \land t \Leftrightarrow t)$

$\vdash \forall\ t\bullet\ (T \lor t) \land (t \lor T) \land (F \lor t \Leftrightarrow t) \land (t \lor F \Leftrightarrow t) \land (t \lor t \Leftrightarrow t)$

Errors As the failure of the conversion argument.

SML

val **all_⇒_intro** :THM −> THM;

Description Discharge all members of assumption list using *⇒_intro*.

Rule

$$\frac{\{t1,\ ...,\ tn\} \vdash t}{\vdash t1 \Rightarrow ... \Rightarrow tn \Rightarrow t} \qquad all\_{\Rightarrow}\_intro$$

SML

val **all_∀_arb_elim** : THM −> THM;

Description Specialise all the quantifiers of a possibly universally quantified theorem with a machine generated variables or variable structures.

Rule

$$\frac{\Gamma \vdash \forall\ vs1[x1,y1,...]\ vs2[x2,y2,...]\ ...\bullet\ p[x1,y1,....x2,y2,...]}{\Gamma \vdash p[x1',y1',...,x2',y2',...]} \qquad \forall\_arb\_elim$$

where x_i', y_i', etc, are not variables (free or bound) in p or Γ, created by *gen_vars*(q.v).

See Also *all_∀_elim*

SML

val **all_∀_elim** : THM −> THM;

Description Specialises all the outer universal quantifications in a theorem:

Rule

$$\frac{\Gamma \vdash \forall\ x1\ ...\ xn\bullet\ t[x1,\ ...,\ xn]}{\Gamma \vdash t[x1',\ ...,\ xn']} \qquad all\_\forall\_elim$$

where $x1', \ldots, xn'$ are renamed from $x1, \ldots, xn$ as necessary to avoid name clashes with free variables in the assumption list.

See Also *all_∀_arb_elim* which is faster, though the results are slightly opaque. *list_∀_elim*.

SML

val **all_∀_intro** : THM −> THM;

Description Generalises all the free variables (other than those in the assumption list) in a theorem:

Rule

$$\frac{\Gamma \vdash t}{\Gamma \vdash \forall\ x1\ \ldots\ xn \bullet t} \quad all\_\forall\_intro$$

where $x1, \ldots, xn$ are all the free variables of t. The function introduces variables in their order of occurrence, so:

Example

$all\_\forall\_intro\ (\vdash a \lor b) = \vdash \forall\ a\ b \bullet a \lor b$

SML

val **all_∀_uncurry_conv** : $CONV$;

Description Apply $\forall\_uncurry\_conv$ (q.v) to the outer universal quantifications of a term, flattening those binders.

Conversion

$$\frac{}{\begin{array}{l}\Gamma \vdash (\forall\ vs1[x,y,\ldots]\ vs2[x,y,\ldots]\ \ldots\bullet \\ \quad f[x1,y1,\ldots,x2,y2,\ldots]) \\ \quad = (\forall\ x1\ y1\ \ldots\ x2\ y2\ \ldots\bullet \\ \quad f[x1,y1,\ldots,x2,y2,\ldots])\end{array}} \quad \begin{array}{l}all\_\forall\_uncurry\_conv \\ \ulcorner\forall\ vs1[x1,y1,\ldots]\ vs2[x2,y2,\ldots]\ \ldots\bullet \\ \quad f[x1,y1,\ldots,x2,y2,\ldots]\urcorner\end{array}$$

where the $vs\_i[x\_i, y\_i, \ldots]$ are variable structures at least one of which must not be a simple variable, built from variables $x\_i, y\_i, \ldots,$

Errors

27041 ?0 is not of the form: $\ulcorner\forall\ \ldots\ (x,y)\ \ldots\bullet f\urcorner$

SML

val **all_∃_uncurry_conv** : $CONV$;

Description Apply $\exists\_uncurry\_conv$ (q.v) to the outer existential quantifications of a term, flattening those binders.

Conversion

$$\frac{}{\begin{array}{l}\Gamma \vdash (\exists\ vs1[x,y,\ldots]\ vs2[x,y,\ldots]\bullet \\ \quad f[x1,y1,\ldots,x2,y2,\ldots]) \\ \quad = (\exists\ x1\ y1\ \ldots\ x2\ y2\ \ldots\bullet \\ \quad f[x1,y1,\ldots,x2,y2,\ldots])\end{array}} \quad \begin{array}{l}all\_\exists\_uncurry\_conv \\ \ulcorner\exists\ vs1[x1,y1,\ldots]\ vs2[x2,y2,\ldots]\ \ldots\bullet \\ f[x1,y1,\ldots,x2,y2,\ldots]\urcorner\end{array}$$

where the $vs[x, y, \ldots]$ are variable structures with variables $x, y, \ldots,$ at least one of which must not be a simple variable.

See Also $all\_\forall\_uncurry\_conv$

Errors

27048 ?0 is not of the form: $\ulcorner\exists\ \ldots\ (x,y)\ \ldots\bullet f\urcorner$

SML
```
val all_β_conv : CONV;
```

Description A conversion to eliminate all instances of β redexes, including paired abstraction redexes, in a term, regardless of nesting, or even that the β redex was created as the result of an earlier reduction in the conversion's evaluation.

Rule

$$\frac{}{\vdash t = t'} \qquad \begin{array}{l} all\_\beta\_conv \\ \ulcorner t \urcorner \end{array}$$

t' is t with all β redexes reduced.

Uses This uses an optimised term traversal algorithm, superior in speed to the general term traversal algorithms used with conversions, and should be used in preference to them and *β_conv*.

See Also *all_simple_β_conv* which only handles simple β-redexes, but does a faster traversal if that is all that is required. *all_β_rule*.

Errors

27049 ?0 contains no β−redexes

SML
```
val all_β_rule : THM -> THM;
```

Description Eliminate all instances of β redexes, including paired abstraction redexes, in the conclusion of a theorem, regardless of nesting, or even that the β redex was created as the result of an earlier reduction in the rule's evaluation.

Rule

$$\frac{\Gamma \vdash t}{\Gamma \vdash t'} \qquad all\_\beta\_rule$$

t' is t with all β-redexes reduced.

See Also *all_β_conv* for the conversion. *all_simple_β_rule* which only handles simple β-redexes, but does a faster traversal if that is all that is required.

Errors

27049 ?0 contains no β−redexes

SML
```
val AND_OR_C : (CONV * CONV) -> CONV;
```

Description *c1 AND_OR_C c2* will succeed if it can apply one or both of *c1* or *c2*. If it cannot compose the results of applying both conversions successfully (indicating an ill-formed conversion result) it will return the result of the first conversion application.

See Also *THEN_TRY_C*, *ORELSE_C*, *THEN_C*

Errors As the failure message of the second conversion (implying that neither conversion was successfully applied).

SML
```
val app_arg_rule : TERM -> THM -> THM;
```

Description Apply both sides of an equational theorem to an argument.

Rule

$$\frac{\Gamma \vdash f = g}{\Gamma \vdash f\ x = g\ x} \qquad \begin{array}{l} app\_arg\_rule \\ \ulcorner x \urcorner \end{array}$$

Errors

6020 *?0 is not of the form:* ‘$\Gamma \vdash t1 = t2$‘
7025 *Sides of equation may not be applied to term*

SML
```
val APP_C : (CONV * CONV) -> CONV;
```

Description Apply one conversion to the operator of a combination, and a second to the operand.

Rule

$$\frac{}{\vdash f\ a = f'\ a'} \qquad \begin{array}{l} APP\_C \\ (c1 : CONV, \\ c2 : CONV) \\ \ulcorner f\ a \urcorner \end{array}$$

where $c1\ f$ gives ‘$\vdash f = f'$‘, and $c2\ f$ gives ‘$\vdash a = a'$‘.

Errors

3010 *?0 is not of form:* $\ulcorner t1\ t2 \urcorner$
7110 *Results of conversions, ?0 and ?1, ill−formed or cannot be combined*

Also as the failure of the conversions.

SML
```
val app_fun_rule : TERM -> THM -> THM ;
```

Description Apply a function to both sides of an equational theorem.

Rule

$$\frac{\Gamma \vdash a = b}{\Gamma \vdash f\ a = f\ b} \qquad \begin{array}{l} app\_fun\_rule \\ \ulcorner f \urcorner \end{array}$$

Errors

6020 *?0 is not of the form:* ‘$\Gamma \vdash t1 = t2$‘
7024 *?0 may not be applied to each side of equation*

SML
```
val app_if_conv : CONV;
```

Description Move a function application into a conditional.

Conversion

$$\frac{}{\begin{array}{l}\vdash f(if\ a\ then\ b\ else\ c) = \\ \quad (if\ a\ then\ f\ b\ else\ f\ c)\end{array}} \qquad \begin{array}{l} app\_if\_conv \\ \ulcorner f(if\ a\ then\ b\ else\ c) \urcorner \end{array}$$

Errors

7098 *?0 is not of the form:* $\ulcorner f(if\ a\ then\ b\ else\ c) \urcorner$

SML
```
val asm_elim : TERM -> THM -> THM -> THM;
```

Description Eliminate an assumption with reference to contradictory assumption lists.

Rule

$$\frac{\Gamma 1,\ a' \vdash t;\ \Gamma 2,\ \neg a''\ \vdash t'}{\Gamma 1\ \cup\ \Gamma 2 \vdash t} \qquad \begin{array}{l} asm\_elim \\ \ulcorner a \urcorner \end{array}$$

where a, a' and a'', as well as t and t' are α-convertible. Actually, the assumptions don't have to be present for the function to succeed.

Errors

3031 *?0 is not of type* ⌜:*BOOL*⌝
7029 *?0 and ?1 are not of the form:'Γ1, aa ⊢ t' and 'Γ2, ¬aaa ⊢ ta'*
where ⌜*t*⌝ *and* ⌜*ta*⌝ *are α−convertible*

SML
```
val asm_inst_term_rule : (TERM * TERM) list -> THM -> THM;
```

Description Parallel instantiation of term variables within a theorem's conclusion and assumptions to some other values.

Rule

$$\frac{\Gamma \vdash t[x1,\ ...,\ xn]}{\Gamma' \vdash t[t1,\ ...,\ tn]} \qquad \begin{array}{l} asm\_inst\_term\_rule \\ [...,\ (\ulcorner ti \urcorner,\ \ulcorner xi \urcorner),\ ...] \end{array}$$

See Also *inst_term_rule*

Errors

3007 *?0 is not a term variable*
6027 *Types of element (?0, ?1) in term association list differ*

SML
```
val asm_inst_type_rule : (TYPE * TYPE) list -> THM -> THM;
```

Description Parallel instantiation of some of the type variables of both the conclusion and assumptions of a theorem.

Rule

$$\frac{\Gamma \vdash t[tv1,...tvn]}{\Gamma' \vdash t[\sigma 1,...\sigma n]} \qquad \begin{array}{l} asm\_inst\_type\_rule \\ [(\sigma 1,\ tv1),\ ...,\ (\sigma n,tvn)] \end{array}$$

asm_inst_type_rule talist thm will instantiate each type variable in *talist* with its associated type. It will decorate free variables that would become identified with other variables by their types becoming the same and the names originally being the same. α-convertible duplicate assumptions will be eliminated.

See Also *inst_type_rule*

Errors

3019 *?0 is not a type variable*

SML
```
val asm_intro : TERM -> THM -> THM;
```

Description Introduce a new assumption to an existing theorem.

Rule

$$\frac{\Gamma \vdash t2}{\Gamma \cup \{t1\} \vdash t2} \qquad \begin{array}{l} asm\_intro \\ \ulcorner t1 \urcorner \end{array}$$

Errors

3031 *?0 is not of type* ⌜:*BOOL*⌝

SML
```
val asm_rule : TERM -> THM;
```

Description "A term is true on the assumption that it is true."

Rule

$$\frac{}{t \vdash t} \qquad \begin{array}{l} asm\_rule \\ \ulcorner t \urcorner \end{array}$$

A primitive inference rule.

Errors

3031	*?0 is not of type* ⌜*:BOOL*⌝

SML
```
val BINDER_C : CONV -> CONV;
```

Description Apply a conversion to the body of a binder term:

Rule

$$\frac{}{\vdash (B\ x\bullet\ p[x]) = (B\ x\bullet\ pa[x])} \qquad \begin{array}{l} BINDER\_C \\ (c : CONV) \\ \ulcorner B\ x\bullet\ p \urcorner \end{array}$$

where $c\ p[x]$ gives '$\vdash p[x] = pa[x]$', and B is a binder.

Errors

27035	*?0 is not of the form:* ⌜*B x• p[x]*⌝ *where* ⌜*B*⌝ *is a binder and* ⌜*x*⌝ *a varstruct*
7104	*Result of conversion, ?0, ill−formed*

Also as the failure of the conversion.

SML
```
val CHANGED_C : CONV -> CONV;
```

Description Applies a conversion, and fails if either the conversion fails, has ill-formed results in certain ways, or it causes no change. Even α-convertible changes count as a change for this purpose.

Errors

7032	*Conversion failed to cause a change*
7104	*Result of conversion, ?0, ill−formed*

It may also fail with the error message of the conversion argument.

SML
```
val char_conv : CONV;
```

Description This function defines the character literal constants, by giving a relationship between character literal constants and their ASCII code (derived by the Standard ML function *ord*). A character literal is indicated by the constant's name starting with single backquote (‘), being a single other character, as well as being of type *CHAR*.

Rule

$$\frac{}{\begin{array}{c}\vdash {}_{\mathrm{ML}}\ulcorner(mk\_char("c"))\urcorner = \\ AbsChar\ {}_{\mathrm{ML}}\ulcorner ord\ "c"\urcorner\end{array}} \qquad \begin{array}{l} char\_conv \\ (mk\_char("c")) \end{array}$$

A primitive inference rule(axiom schemata).

See Also *mk_char*

Errors

3024	*?0 is not a character literal*

SML
$|$*val* **COND_C** : (*TERM* −> *bool*) −> *CONV* −> *CONV* −> *CONV*;

Description *COND_C pred cnv1 cnv2 tm* will be, if the term predicate *pred* applied to *tm* is true, then *cnv1 tm* and otherwise the *cnv2 tm*.

Errors As the failure of the predicate or either conversion.

SML
$|$*val* **cond_thm** : *THM*;

Description A convenient variant of the definition of the conditional.

Theorem

$$\frac{}{\vdash \forall\ a\ t1\ t2\bullet\ (if\ a\ then\ t1\ else\ t2) = (\epsilon\ x\bullet\ ((a \Leftrightarrow T) \Rightarrow x = t1) \land ((a \Leftrightarrow F) \Rightarrow x = t2))} \quad cond\_thm$$

SML
$|$*val* **contr_rule** : *TERM* −> *THM* −> *THM*;

Description Intuitionistic contradiction rule:

Rule

$$\frac{\Gamma \vdash F}{\Gamma \vdash t} \quad \begin{array}{l} contr\_rule \\ \ulcorner t \urcorner \end{array}$$

Errors
$|$*7001 ?0 is not of form: ‘Γ ⊢ F‘*
$|$*3031 ?0 is not of type* $\ulcorner$*:BOOL*$\urcorner$

SML
$|$*val* **conv_rule** : *CONV* −> *THM* −> *THM*;

Description Apply a conversion to the conclusion of a theorem, and do ⇔ modus ponens between the original theorem and the result of the conversion

Rule

$$\frac{\Gamma 1 \vdash t}{\Gamma 1 \cup \Gamma 2 \vdash t'} \quad \begin{array}{l} conv\_rule \\ (c : CONV) \end{array}$$

where $c\ t$ gives $\Gamma 2 \vdash t \Leftrightarrow t'$.

Errors
$|$*7104 Result of conversion, ?0, ill−formed*

Also as the failure of the conversion upon the conclusion of the theorem.

SML
$|$*val* **cthm_eqn_cxt** : *CANON* −> *THM* −> *EQN_CXT*;

Description This function applies a canonicalisation (see *CANON*) to a theorem, and then attempts to convert each of the list of resulting theorems into an equational context entry using *thm_eqn_cxt* (q.v.). The results are composed into an equational context (which is only a Standard ML list of equational context entries). Canoncalised theorems that cannot be converted by *thm_eqn_cxt* will be discarded.

SML

```
val c_contr_rule : TERM -> THM -> THM;
```

Description Classical contradiction rule:

Rule

$$\frac{\Gamma,\ \neg t' \vdash F}{\Gamma \vdash t} \qquad \begin{array}{l} c\_contr\_rule \\ \ulcorner t \urcorner \end{array}$$

Note that the argument is the unnegated form of what must be present in the assumption list for success. Works up to α-conversion.

Errors

7001	*?0 is not of form: '$\Gamma \vdash F$'*
3031	*?0 is not of type ⌜:BOOL⌝*
7003	*Negation of ?0 is not in assumption list*

SML

```
val disch_rule : TERM -> THM -> THM;
```

Description Prove an implicative theorem, removing, if α-convertibly present, the antecedent of the implication from the assumption list, and failing if it is not present.

Rule

$$\frac{\Gamma,\ t1' \vdash t2}{\Gamma \vdash t1 \Rightarrow t2} \qquad \begin{array}{l} disch\_rule \\ \ulcorner t1 \urcorner \end{array}$$

See Also *⇒_intro* (which does not fail if term not in assumption list)

Errors

7031	*?0 not α−convertibly present in assumption list*

SML

```
val eq_match_conv1 : THM -> CONV ;
```

Description This matches the LHS of an universally quantified (simple or by varstruct) equational theorem to a term, instantiating the RHS accordingly. The conversion will only instantiate its universal quantifications, and type variables not found within the assumptions, not its free term variables.

Conversion

$$\frac{}{\Gamma \vdash t = v[t1,...,tn]} \qquad \begin{array}{l} eq\_match\_conv1 \\ (\Gamma \vdash \forall\ x1\ ...\ xn\bullet\ u[x1,...,xn] = \\ \quad v[x1,...,xn]) \\ \ulcorner t \urcorner \end{array}$$

where ⌜u[t1,...,tn]⌝ is α-convertible to ⌜t⌝. If there are free variables on the RHS of the supplied equational theorem (when stripped of all universal quantification) they will be renamed as necessary to avoid identification with any variables in t.

This conversion may be partially evaluated with only its theorem argument.

Uses In producing a limited rewriting facility, that only instantiates explicitly identified variables.

Errors

27003	*?0 is not of the form '$\Gamma \vdash \forall\ x1\ ...\ xn\bullet\ u = v$' where ⌜xi⌝ are varstructs*
7076	*Could not match term ?0 to LHS of theorem ?1*

SML

```
val eq_match_conv : THM -> CONV ;
```

Description This matches the LHS of an equational theorem to a term, instantiating the RHS accordingly. The equational theorem may be partially or fully universally quantified (simple or by varstruct), without affecting the result of the conversion.

Conversion

$$\frac{}{\Gamma \vdash t = v'} \qquad \begin{array}{l} eq\_match\_conv \\ (\Gamma \vdash \forall \ldots \bullet\ u = v) \\ \ulcorner t \urcorner \end{array}$$

where v' is the result of applying to v the instantiation rules required to match u to t (including both term and type instantiation). If there are free variables on the RHS of the supplied equational theorem (when stripped of all universal quantification) they will be renamed as necessary to avoid identification with any variables in t.

This conversion may be partially evaluated with only its theorem argument.

See Also *eq_match_conv1*

Errors

7044	*Cannot match ?0 and ?1*

SML

val **eq_rewrite_thm** : THM
val **⇔_rewrite_thm** : THM
val **¬_rewrite_thm** : THM
val **∧_rewrite_thm** : THM
val **∨_rewrite_thm** : THM
val **⇒_rewrite_thm** : THM
val **if_rewrite_thm** : THM
val **∀_rewrite_thm** : THM
val **∃_rewrite_thm** : THM
val **β_rewrite_thm** : THM

Description These are some of the default list of theorems used by those rewriting rules, conversions and tactics whose names do not begin with '*pure_*':

eq_rewrite_thm $\vdash \forall\ x \bullet (x = x) \Leftrightarrow T$

⇔_rewrite_thm $\vdash \forall\ t \bullet ((T \Leftrightarrow t) = t) \land ((t \Leftrightarrow T) = t) \land$
$((F \Leftrightarrow t) = (\neg\ t)) \land (t \Leftrightarrow F) = (\neg\ t)$

¬_rewrite_thm $\vdash \forall\ t \bullet (\neg\neg t) = t \land ((\neg\ T) = F) \land (\neg\ F) = T$

∧_rewrite_thm $\vdash \forall\ t \bullet ((T \land t) = t) \land ((t \land T) = t) \land$
$(\neg\ (F \land t)) \land (\neg\ (t \land F)) \land (t \land t) = t$

∨_rewrite_thm $\vdash \forall\ t \bullet (T \lor t) \land (t \lor T) \land ((F \lor t) = t) \land ((t \lor F) = t) \land (t \lor t) = t$

⇒_rewrite_thm $\vdash \forall\ t \bullet ((T \Rightarrow t) = t) \land ((F \Rightarrow t) = T) \land ((t \Rightarrow T) = T) \land ((t \Rightarrow t) = T)$
$\land\ (t \Rightarrow F) = (\neg\ t)$

if_rewrite_thm $\vdash \forall\ t1\ t2{:}'a \bullet ((if\ T\ then\ t1\ else\ t2) = t1) \land (if\ F\ then\ t1\ else\ t2) = t2$

∀_rewrite_thm $\vdash \forall\ t \bullet (\forall\ x \bullet t) = t$
∃_rewrite_thm $\vdash \forall\ t \bullet (\exists\ x \bullet t) = t$

β_rewrite_thm $\vdash \forall\ t1{:}'a;\ t2{:}'b \bullet ((\lambda\ x \bullet t1) t2) = t1$

The theorems are saved in the theory "misc", and given their design in the design for that theory.

See Also *fst_rewrite_thm*, *snd_rewrite_thm*, *fst_snd_rewrite_thm*.

SML

val **eq_sym_conv** : $CONV$;

Description Symmetry of equality:

Rule

$$\frac{}{\vdash (t1 = t2) \Leftrightarrow (t2 = t1)} \quad \begin{array}{l} eq\_sym\_conv \\ \ulcorner t1 = t2 \urcorner \end{array}$$

See Also *eq_sym_rule*

Errors

3014 *?0 is not of form*: $\ulcorner t = u \urcorner$

SML
```
val eq_sym_rule : THM -> THM;
```

Description Symmetry of equality:

Rule

$$\frac{\Gamma \vdash t1 = t2}{\Gamma \vdash t2 = t1} \qquad eq\_sym\_rule$$

A built-in inference rule.

See Also *eq_sym_conv*

Errors

6020	*?0 is not of the form: '$\Gamma \vdash t1 = t2$'*

SML
```
val eq_trans_rule : THM -> THM -> THM;
```

Description Transitivity of equality:

Rule

$$\frac{\Gamma 1 \vdash t1 = t2;\ \Gamma 2 \vdash t2' = t3}{\Gamma 1 \cup \Gamma 2 \vdash t1 = t3} \qquad eq\_trans\_rule$$

where $t2$ and $t2'$ are α convertible. A built-in inference rule.

Errors

6020	*?0 is not of the form: '$\Gamma \vdash t1 = t2$'*
6022	*?0 and ?1 are not of the form: '$\Gamma 1 \vdash t1 = t2$' and '$\Gamma 2 \vdash t2a = t3$' where $\ulcorner t2 \urcorner$ and $\ulcorner t2a \urcorner$ are α−convertible*

SML
```
val EVERY_CAN : CANON list -> CANON
```

Description *EVERY_CAN* is a canonicalisation function combinator which combines the elements of its argument using *THEN_CAN*:

```
EVERY_CAN [can1, can2, ...] = can1 THEN_CAN can2 THEN_CAN ...
```

See Also *CANON*

SML
```
val EVERY_C : CONV list -> CONV;
```

Description Apply each conversion in the list, in the sequence given.

See Also *THEN_C*(which this function iterates)

Errors

7103	*List may not be empty*

or as the failure of any constituent conversion, or as *THEN_C*.

SML
```
val ext_rule : THM -> THM;
```

Description Extensionality of functions in ICL HOL.

Rule

$$\frac{\Gamma \vdash f = g}{\Gamma \vdash \forall\ x \bullet f\ x = g\ x} \quad ext\_rule$$

where x is a machine-generated variable of appropriate type, not found free in the equational theorem.

Errors

6020 ?0 is not of the form: ‘Γ ⊢ t1 = t2‘
7026 ?0 is not an equation of functions

SML
```
val fail_canon : CANON
```

Description This is a canonicalisation function which always fails. It is the identity for *ORELSE_CAN*.

See Also *CANON*

Errors

26201 Failed as requested

SML
```
val fail_conv : CONV;
```

Description This conversion always fails.

Errors

7061 Failed as requested

SML
```
val fail_with_canon : string -> int -> (unit -> string) list -> CANON
```

Description This is a canonicalisation function which always fails by passing its arguments to *fail* (q.v.).

See Also *fail_can*

SML
```
val fail_with_conv : string -> CONV;
```

Description This conversion always fails, with the error message being its string argument.

Errors

7075 ?0

SML
```
val FIRST_CAN : CANON list -> CANON
```

Description *FIRST_CAN* is a canonicalisation function combinator which combines the elements of its argument using *ORELSE_CAN*:

```
FIRST_CAN [can1, can2, ...] = can1 ORELSE_CAN can2 ORELSE_CAN ...
```

See Also *CANON*

Errors

26202 the list of canonicalisation functions is empty

SML
```
val FIRST_C : CONV list -> CONV;
```

Description Attempt to apply each conversion in the list, in the sequence given, until one succeeds, or all fail.

See Also *ORELSE_C*(which this function iterates)

Errors
```
7103  List may not be empty
```

or as the failure of the last conversion.

SML
```
val FORWARD_CHAIN_CAN : CANON list -> CANON;
val FC_CAN : CANON list -> CANON;
```

Description *FORWARD_CHAIN_CAN*, which has the alias *FC_CAN*, is a parameterised variant of *fc_canon*. Given a list of canonicalisation functions *cans*, *FC_CAN cans* behaves as *fc_canon* would do if the line

$\vdash A \quad\quad \to \quad\quad FIRST\_CAN\ cans\ A$

were inserted at the beginning of the table of transformations given in the description of *fc_canon*.

For example, *fc_canon1*, q.v., is the same as:

```
        FC_CAN ((fn (x, y) => [x,y]) o ⇔_elim);
```

Uses In tactic programming, or, occasionally interactively, typically in circumstances where neither *fc_canon* nor *fc_canon1* is able to generate enough implications.

SML

```
val forward_chain_canon : THM -> THM list;
val fc_canon : THM -> THM list;
val forward_chain_canon1 : THM -> THM list;
val fc_canon1 : THM -> THM list;
```

Description *forward_chain_canon* is a canonicalisation function which uses a theorem to generate a list of implications. (*fc_canon* is an alias for *forward_chain_canon*.) It may be used for constructing rules and tactics in conjunction with *forward_chain_rule*. An example of such a tactic is *forward_chain_tac*. *forward_chain_canon1*, which has alias *fc_canon1*, is just like *fc_canon* except for its treatment of bi-implications. The effects of *fc_canon* and *fc_canon1* are shown schematically in the following table (which only shows assumptions relevant to the process):

$\vdash A \land B$	$\rightarrow$	$\vdash A$; $\vdash B$	
$\vdash \forall x \bullet A$	$\rightarrow$	$\vdash A[x'/x]$	
$\vdash A \land B \Rightarrow C$	$\rightarrow$	$map\ (\Rightarrow\_intro\ (st\ulcorner A\urcorner))\ (xf(\ st\ulcorner A\urcorner \vdash B \Rightarrow C\ ))$	
$\vdash A \lor B \Rightarrow C$	$\rightarrow$	$xf(\ \vdash (A \Rightarrow C) \land (B \Rightarrow C)\ )$	
$\vdash (\exists x \bullet A) \Rightarrow C$	$\rightarrow$	$map\ (\forall\_intro\ \ulcorner x'\urcorner)\ (xf(\ \vdash A[x'/x] \Rightarrow B\ ))$	
$A \vdash A \Rightarrow B$	$\rightarrow$	$A \vdash B$	
$\vdash T \Rightarrow B$	$\rightarrow$	$\vdash B$	
$A \vdash \neg A \Rightarrow B$	$\rightarrow$	(* *discarded* *)	
$\vdash F \Rightarrow B$	$\rightarrow$	(* *discarded* *)	
$\vdash A \Rightarrow B$	$\rightarrow$	$map\ (\Rightarrow\_intro\ (st\ulcorner A\urcorner))\ (xf(\ st\ulcorner A\urcorner \vdash B\ ))$	
$\vdash A \Leftrightarrow B$	$\rightarrow$	$\vdash A \Rightarrow B$	(* *fc_canon* *)
$\vdash A \Leftrightarrow B$	$\rightarrow$	$\vdash A \Rightarrow B; \vdash B \Rightarrow A$	(* *fc_canon1* *)
$\vdash T$	$\rightarrow$	(* *discarded* *)	
$\vdash A$	$\rightarrow$	$\vdash sc\ulcorner A\urcorner$	
$\vdash A$	$\rightarrow$	$\vdash \neg A \Rightarrow F$	

The intention here is that is that the first applicable transformation is applied repeatedly until no further change is possible. The resulting theorems are then universally quantified over all of the free variables in their conclusions which were not free in the original theorem. In the table, *st* and *sc* stand for attempts to apply the theorem and conclusion stripping conversions in the current proof context (as returned by *current_ad_st_conv* and *current_ad_sc_conv*). If the stripping conversions fail then *st* and *sc* have no effect. x' denotes a variable name derived from x and chosen to avoid variable capture problems. *xf* stands for a nested recursive application of the transformation process.

In the transformations involving $\Rightarrow\_intro$ the implication is only introduced if the antecedent is in the assumptions. So, for example, $A \Rightarrow B \Rightarrow A \Rightarrow C$ is transformed into $B \Rightarrow A \Rightarrow C$. The transformation for $A \Rightarrow B$ is only applied if it changes the theorem, and the last of the transformations is only applied if A is neither an implication nor F.

The asymmetry in the rules is deliberate. E.g., they derive $A \Rightarrow B \Rightarrow C$ from $A \land B \Rightarrow C$, but not $B \Rightarrow A \Rightarrow C$. This is intended to give slightly finer control and to result in less duplication of results in the intended application in *forward_chain_tac*(q.v.).

See Also *forward_chain_rule*, *forward_chain_tac*, *FC_CAN*

SML

```
val forward_chain_rule : THM list -> THM list -> THM list;
val fc_rule : THM list -> THM list -> THM list;
```

Description This is a rule which uses a list of possibly universally quantified implications and a list of other theorems to infer new theorems, using the matching modus ponens rule from the proof context, if present, or *⇒_match_mp_rule2*if *current_ad_mmp_rule*() returns *Nil*. (*fc_rule* is an alias for *forward_chain_rule*.) *fc_rule imps ants* returns the list of all theorems which may be derived by applying the matching modus ponens rule to a theorem from *imps* and one from *ants*. As a special case, if any theorem to be returned is determined to have ⌜F⌝ as its conclusion, the first such found wil be returned as a singleton list. In order to work well in conjunction with *fc_canon* and *fc_tac* the theorems returned by the matching modus ponens rule are transformed as follows:

1. Theorems of the form: $\vdash \forall\ x_1\ ...\bullet\ t_1 \Rightarrow t_2 \Rightarrow ... \Rightarrow \neg t_k \Rightarrow F$ have their final implication changed to t_k.

2. Theorems of the form: $\vdash \forall\ x_1\ ...\bullet\ t_1 \Rightarrow t_2 \Rightarrow ... \Rightarrow t_k \Rightarrow F$ have their final implication changed to $\Rightarrow\backslash\neg t_k$.

3. All theorems are universally quantified over all the variables which appear free in their conclusions but not in their assumptions (using *all_∀_intro*).

Note that when the matching modus ponens rule is either *⇒_match_mp_rule2* or *⇒_match_mp_rule1*, there is some control over the number of results generated, since variables which appear free in *imps* are not considered as candidates for instantiation.

The rule does not check that the theorems in its first argument are (possible universally) quantified implications.

See Also *forward_chain_tac*, *forward_chain_canon*.

SML

```
val FORWARD_CHAIN_⇔_CAN : CANON list -> CANON;
val FC_⇔_CAN : CANON list -> CANON;
```

Description These are just like *FORWARD_CHAIN_CAN*, q.v., except that they do *not* break up bi-implications. Thus, given a list of canonicalisation functions *cans*, *FC_⇔_CAN cans* behaves as *fc_canon* would do if the line

$\vdash A \quad\quad \rightarrow \quad\quad FIRST\_CAN\ cans\ A$

were inserted at the beginning of the table of transformations given in the description of *fc_canon*and all transformations (including those coming from the proof context) that eliminate bi-implications were suppressed.

Uses In tactic programming, or, occasionally interactively, typically in circumstances where *fc_⇔_canon* is not able to generate enough implications.

SML
```
val forward_chain_⇔_canon : THM -> THM list;
val fc_⇔_canon : THM -> THM list;
```

Description *forward_chain_⇔_canon* is a canonicalisation function very similar to *forward_chain_canon*, q.v. The difference is that *forward_chain_⇔_canon* suppresses all transformations which break up bi-implications. It is intended for use in situations where a bi-implication is to be used as a conditional rewrite rule.

For example, the tactic *ALL_ASM_FC_T1 fc_⇔_canon rewrite_tac* [] can instantiate an assumption of the form $\forall x1\ x2\ ...\bullet\ A \Rightarrow B \Rightarrow (C \Leftrightarrow D)$ and use the result to rewrite instances of C.

See Also *FC_T1*, *ALL_FC_T1* etc.

SML
```
val f_thm : THM;
```

Description "Not False" is true.

Theorem

$$\frac{}{\vdash \neg\ F} \quad f\_thm$$

SML
```
val id_canon : CANON
```

Description This is the identity for the canonicalisation function combinator *THEN_CAN*:

```
id_canon thm = [thm]
```

See Also *CANON*

SML
```
val id_conv : CONV;
```

Description This is an alias for *refl_conv*, reflecting the fact that *refl_conv* is the identity for the conversional *THEN_C*.

Errors
```
7061   Failed as requested
```

SML
```
val if_app_conv : CONV;
```

Description Move a function application out of a conditional.

Conversion

$$\frac{}{\vdash (if\ a\ then\ f\ b\ else\ f'\ c) = f(if\ a\ then\ b\ else\ c)} \quad \begin{array}{l} if\_app\_conv \\ \ulcorner(if\ a\ then\ f\ b\ else\ f\ c)\urcorner \end{array}$$

where f and f' are α-convertible, and f is used on the RHS of the resulting equational theorem

Errors
```
7037   ?0 is not of the form: ⌜if a then (f b) else (g c)⌝
7038   ?0 is not of the form: ⌜if a then (f b) else (fa c)⌝
       where ⌜f⌝ and ⌜fa⌝ are α−convertible
```

SML
```
val if_else_elim : THM -> THM;
```

Description Give the dependence of the *else* branch of a conditional upon the condition.

Rule

$$\frac{\Gamma \vdash \mathit{if}\ \ tc\ \ \mathit{then}\ \ tt\ \ \mathit{else}\ \ te}{\Gamma \vdash \neg\ tc \Rightarrow te} \qquad \mathit{if\_else\_elim}$$

Errors

7012 ?0 is not of the form: ‘Γ ⊢ if tc then tt else te‘

SML
```
val if_intro : TERM -> THM -> THM -> THM;
```

Description Introduce a conditional, based on the assumptions of two theorems.

Rule

$$\frac{\Gamma 1,\ a \vdash tt\ ;\ \ \Gamma 2,\ \neg a' \vdash et}{\Gamma 1 \cup \Gamma 2 \vdash \mathit{if}\ \ a\ \ \mathit{then}\ \ tt\ \ \mathit{else}\ \ et} \qquad \begin{array}{l}\mathit{if\_intro}\\ \ulcorner a \urcorner\end{array}$$

where a and a' are α-convertible. Actually, the assumptions may be missing, and the rule still works.

Example
```
(⊢ x = tt), (⊢ x = te)                      (∗ hypothesis ∗)
⊢ if a then (x = tt) else (x = te)   (∗ if_intro ⌜a⌝ ∗)
⊢ x = if a then tt else te              (∗ if_fun_rule ∗)
```

Errors

3031 ?0 is not of type ⌜:BOOL⌝

SML
```
val if_then_elim : THM -> THM;
```

Description Give the dependence of the *then* branch of a conditional upon the condition.

Rule

$$\frac{\Gamma \vdash \mathit{if}\ \ tc\ \ \mathit{then}\ \ tt\ \ \mathit{else}\ \ te}{\Gamma \vdash tc \Rightarrow tt} \qquad \mathit{if\_then\_elim}$$

Errors

7012 ?0 is not of the form: ‘Γ ⊢ if tc then tt else te‘

SML

val **initial_rw_canon** : $CANON$;

Description This is the initial rewrite canonicalisation function, defined as

```
val initial_rw_canon =
        REWRITE_CAN
        (REPEAT_CAN(FIRST_CAN [
        simple_∀_rewrite_canon,
        ∧_rewrite_canon,
        simple_¬_rewrite_canon,
        f_rewrite_canon,
        ⇔_t_rewrite_canon]));
```

This is the repeated application of the first applicable operation in the following list:

1. stripping universal quantifiers;
2. dividing conjunctive theorems into their conjuncts;
3. changing $\vdash\neg(t1\vee t2)$ to $\neg t1\wedge\neg t2$;
4. changing $\vdash\neg\exists x\bullet t$ to $\forall x\bullet\neg t$;
5. changing $\vdash\neg\neg t$ to t;
6. changing $\vdash\neg t$ to $t\Leftrightarrow F$;
7. changing $\vdash F$ to $\vdash \forall\ x \bullet x$;
8. if none of the above apply, changing $\vdash t$ to $\vdash t = T$.

   Finally, after all this canonicalisation we then universally quantify the resulting theorems in all free variables other than those that were free in the original.

SML

val **inst_term_rule** : $(TERM * TERM)\ list \to THM \to THM$;

Description Parallel instantiation of term variables within a theorem's conclusion to some other values.

Rule

$$\frac{\Gamma \vdash t[x1,\ ...,\ xn]}{\Gamma \vdash t[t1,\ ...,\ tn]} \qquad \begin{array}{l} inst\_term\_rule \\ [...,\ (\ulcorner ti \urcorner,\ \ulcorner xi \urcorner),\ ...] \end{array}$$

A built-in inference rule.

See Also *asm_inst_term_rule*

Errors

3007 ?0 is not a term variable
6027 Types of element (?0, ?1) in term association list differ
6028 Instantiation variable ?0 free in assumption list

SML
```
val inst_type_rule : (TYPE * TYPE) list -> THM -> THM;
```

Description Parallel instantiation of some of the type variables of the conclusion of a theorem.

Rule

$$\frac{\Gamma \vdash t[tyv1,...tyvn]}{\Gamma \vdash t[\sigma 1,...\sigma n]} \qquad \begin{array}{l} inst\_type\_rule \\ [(\sigma 1,\ tyv1),\ ...,\ (\sigma n,tyvn)] \end{array}$$

inst_type_rule talist thm will instantiate each type variable in *talist* with its associated type. It will decorate free variables that would become identified with other variables (both in conclusion and assumptions) by their types becoming the same and the names originally being the same. To instantiate types in the assumption list, see *asm_inst_type_rule*.

A primitive inference rule.

See Also *asm_inst_type_rule* for something that also works on type variables in the assumption list.

Errors

3019	*?0 is not a type variable*
6006	*Trying to instantiate type variable ?0, which occurs in assumption list*

SML
```
val LEFT_C : CONV -> CONV;
```

Description Apply a conversion to the first operand of a binary operator:

Rule

$$\frac{}{\vdash f\ a\ b = f\ a'\ b} \qquad \begin{array}{l} LEFT\_C \\ (c : CONV) \\ \ulcorner f\ a\ b \urcorner \end{array}$$

where $c\ a$ gives $\vdash a = a'$. f may itself be a function application.

Errors

3013	*?0 is not of form:* $\ulcorner f\ a\ b \urcorner$
7104	*Result of conversion, ?0, ill−formed*

Also as the failure of the conversion.

SML
```
val let_conv : CONV;
```

Description Eliminate an outermost *let ... and ... in ...* construct.

Conversion

$$\frac{}{\begin{array}{c} \vdash (let\ vs1[x1,y1,..] = t1 \\ and\ ...\ and\ vsn[xn,yn,..] = tn \\ in\ t[x1,...,xn,...] \\ = t[t1x,...,t1y,...,tnx,tny,...] \end{array}} \qquad \begin{array}{l} let\_conv \\ \ulcorner let\ vs1[x1,y1,..] = \\ \ \ t1\ and\ ...\ vsn[xn,yn,..] = tn \\ \ \ in\ t[x1,...,xn,...]\urcorner \end{array}$$

Where the t_ix is the component of t_i matching x_i when t_i matches $vs\_i[x\_i, y\_i, ..]$.

Errors

4009	*?0 is not of form:* $\ulcorner let\ ...\ in\ ...\urcorner$

SML

val **list_simple_∀_elim** : *TERM list −> THM −> THM*;

Description Generalised ∀ elimination.

Rule

$$\frac{\Gamma \vdash \forall\ x1\ ...\ xn \bullet t[x1,\ ...,\ xn]}{t[t1,\ ...,\ tn]} \qquad \begin{array}{l} list\_simple\_\forall\_elim \\ [\ulcorner t1 \urcorner,\ ...,\ \ulcorner tn \urcorner] \end{array}$$

A built-in inference rule. The instantiation is done simultaneously, rather than by iteration of a single instantiation, which may affect renaming.

See Also *∀_elim*

Errors

3012	*?0 and ?1 do not have the same types*
6018	*?0 is not of the form: '$\Gamma \vdash \forall$ ...xi... • t' where the ⌜xi⌝ are ?1 variables*

SML

val **list_simple_∀_intro** : *TERM list −> THM −> THM*;

Description Generalised simple ∀ introduction.

Rule

$$\frac{\Gamma \vdash t[x1,\ ...,\ xn]}{\Gamma \vdash \forall\ x1\ ...\ xn \bullet t[x1,\ ...,\ xn]} \qquad \begin{array}{l} list\_simple\_\forall\_intro \\ [\ulcorner x1 \urcorner,\ ...,\ \ulcorner xn \urcorner] \end{array}$$

See Also *∀_intro*

Errors Same messages as *simple_∀_intro*.

SML

val **list_simple_∃_intro** : *TERM list −> TERM −> THM −> THM* ;

Description Introduce an iterated existential quantifier by providing a list of witnesses and a theorem asserting that the desired property holds of these witnesses.

Rule

$$\frac{\Gamma \vdash t[t1,t2,...]}{\Gamma \vdash \exists\ x1\ x2... \bullet t[x1,x2,...]} \qquad \begin{array}{l} list\_simple\_\exists\_intro \\ [\ulcorner t1 \urcorner,\ \ulcorner t2 \urcorner,\ ...] \\ \ulcorner \exists\ x1\ x2\ ... \bullet t[x1,x2,...] \urcorner \end{array}$$

Errors

7047	*?0 cannot be matched to conclusion of theorem ?1*

SML

val **list_∧_intro** : *THM list −> THM*;

Description Conjoin a list of theorems.

Rule

$$\frac{[\Gamma 1 \vdash t1,\ ...,\ \Gamma n \vdash tn]}{\Gamma 1 \cup ...\ \Gamma n \vdash t1 \wedge ...\ tn} \qquad list\_\wedge\_intro$$

Errors

7107	*List may not be empty*

SML
```
val list_∀_elim : TERM list -> THM -> THM;
```

Description Generalised ∀ elimination. Specialise a universally quantified theorem with given values, instantiating the types of the theorem as necessary.

Rule

$$\frac{\Gamma \vdash \forall\ x1\ \ldots\ xn \bullet\ t[x1,\ \ldots,\ xn]}{t'[t1,\ \ldots,\ tn]} \qquad \begin{array}{l} list\_\forall\_elim \\ [\ulcorner t1 \urcorner,\ \ldots,\ \ulcorner tn \urcorner] \end{array}$$

where t' is renamed from t to prevent bound variable capture and type instantiated as necessary, the x_i are varstructs, instantiable to the structures of t_i. The values will be expanded using *Fst* and *Snd* as necessary to match the structure of $\ulcorner x \urcorner$.

Note that due to the type instantiation this function is somewhat more that a *fold* of *∀_elim*.

See Also *∀_elim*, *all_∀_elim*.

Errors

27014	*?0 is not of the form: ‘Γ ⊢ ∀ vs1 ... vsi • t‘ where i ≥ ?1*
27015	*?0 is not of the form: ‘Γ ⊢ ∀ vs1 ... vsi • t‘ where the types of the vsi are instantiable to the types of ?1*
27016	*?0 is not of the form: ‘Γ ⊢ ∀ vs1 ... vsi • t‘ where the types of the vsi are instantiable to the types of ?1 without instantiating type variables in the assumptions*

SML
```
val list_∀_intro : TERM list -> THM -> THM;
```

Description Generalised ∀ introduction.

Rule

$$\frac{\Gamma \vdash t[x1,\ \ldots,\ xn]}{\Gamma \vdash \forall\ x1\ \ldots\ xn \bullet\ t[x1,\ \ldots,\ xn]} \qquad \begin{array}{l} list\_\forall\_intro \\ [\ulcorner x1 \urcorner,\ \ldots,\ \ulcorner xn \urcorner] \end{array}$$

See Also *∀_intro*, *all_∀_intro*.

Errors Same messages as *∀_intro*.

SML
```
val MAP_C : CONV -> CONV;
```

Description This traverses a term from its leaves to its root node. It will repeat the application of its conversion argument, until failure, on each subterm encountered en route. At each node the conversion is applied to the sub-term that results from the application of the preceding traversal, not the original. It traverses from left to right, though this should only matter for conversions that work by side-effect. It fails if the conversion applies nowhere within the tree.

Errors

7005	*Conversion fails on term and all its subterms*

SML

val **mk_app_rule** : $THM \;{-}{>}\; THM \;{-}{>}\; THM$;

Description Given two equational theorems, one being between two functions, apply the two functions to the LHS and RHS of the other equation.

Rule

$$\frac{\Gamma 1 \vdash u1 = u2;\ \Gamma 2 \vdash v1 = v2}{\Gamma 1 \cup \Gamma 2 \vdash u1\ v1 = u2\ v2} \qquad mk\_app\_rule$$

The second input theorem or the result may be expressed using ⇔.

A built-in inference rule.

Errors

6020 *?0 is not of the form: ‘Γ ⊢ t1 = t2‘*
6023 *?0 and ?1 are not of the form :‘Γ1 ⊢ u1 = u2‘ and ‘Γ2 ⊢ v1 = v2‘ where ⌜u1⌝ can be functionally applied to ⌜v1⌝*

SML

val **modus_tollens_rule** : $THM \;{-}{>}\; THM \;{-}{>}\; THM$;

Description If the consequent of an implicative theorem is false, then so must be the antecedent (modus tollens).

Rule

$$\frac{\Gamma 1 \vdash t1 \Rightarrow t2;\ \Gamma 2 \vdash \neg t2'}{\Gamma 1 \cup \Gamma 2 \vdash \neg t1} \qquad modus\_tollens\_rule$$

where $t2$ and $t2'$ are α-convertible.

Errors

7040 *?0 is not of the form: ‘Γ ⊢ t1 ⇒ t2‘*
7051 *?0 and ?1 are not of the form: ‘Γ1 ⊢ t1 ⇒ t2‘ and ‘Γ2 ⊢ ¬t2a‘ where ⌜t2⌝ and ⌜t2a⌝ are α−convertible*

SML

val **ONCE_MAP_C** : $CONV \;{-}{>}\; CONV$;

Description This traverses a term from the root node to its leaves, attempting to apply its conversion argument. If it successfully applies the conversion to any subterm then it will not further traverse that subterm, but will still continue on other branches. If it fails to apply its conversion to a leaf, its functionality is equivalent to then applying *refl_conv*. It traverses from left to right, though this should only matter for conversions that work by side-effect. It will fail if the conversion succeeds nowhere in the tree, or if the results of certain conversion applications are ill-formed.

Errors

7005 *Conversion fails on term and all its subterms*

SML
```
val ONCE_MAP_WARN_C : string -> CONV -> CONV;
```

Description This is an equivalent to *ONCE_MAP_C* (q.v.) except that it warns the user if it failed to recompose the theorems from the term it just traversed.

This traverses a term from the root node to its leaves, attempting to apply its conversion argument. If it successfully applies the conversion to any subterm then it will not further traverse that subterm, but will still continue on other branches. If it fails to apply its conversion to a leaf, its functionality is equivalent to then applying *refl_conv*. It traverses from left to right, though this should only matter for conversions that work by side-effect. It will fail if the conversion succeeds nowhere in the tree, or if the results of certain conversion applications are ill-formed.

Errors
```
26001 no rewriting occurred
26003 no successful rewriting occurred, rewriting gave ill-formed results on some subterms
```

It issues the following warning message if at any point it fails to recompose the theorems from the subterm it just traversed, some successful rewriting occurs, and the flag "*illformed_rewrite-_warning*" is true.

Errors
```
26002 rewriting gave ill-formed results on some subterms
```

Errors and warnings are from the area indicated by the string argument.

SML
```
val ORELSE_CAN : (CANON * CANON) -> CANON
```

Description *ORELSE_CAN* is a canonicalisation function combinator written as an infix operator. (*can1 ORELSE_CAN can2*)*thm* is the same *can1 thm* unless evaluation of *can1 thm* fails in which case it is the same as *can2 thm*.

See Also *CANON*

SML
```
val ORELSE_C : (CONV * CONV) -> CONV;
```

Description Attempt to apply one conversion, and if that fails, try the second one.

Rule

$$\frac{}{\Gamma \vdash t = t'} \quad (c1\text{: } CONV)\ ORELSE\_C\ (c2\text{: } CONV)\ \ulcorner t \urcorner$$

where *c1 t* returns $\Gamma \vdash t = t'$, or *c1* fails, and *c2 t* returns $\Gamma \vdash t = t'$.

See Also *FIRST_C* (the iterated version of this function), *THEN_C*, *AND_OR_C*, and *THEN_TRY_C*

Errors As the failure of second conversion, should both conversions fail.

SML

```
val plus_conv : CONV;
```

Description Provides the value of the addition of two numeric literals.

Rule

$$\frac{}{\vdash \ulcorner {}_{\mathrm{ML}}\ulcorner mk\_\mathbb{N}\ m\urcorner + {}_{\mathrm{ML}}\ulcorner mk\_\mathbb{N}\ n\urcorner\urcorner = {}_{\mathrm{ML}}\ulcorner mk\_\mathbb{N}(m + n)\urcorner} \quad \begin{array}{l} plus\_conv \\ \ulcorner {}_{\mathrm{ML}}\ulcorner mk\_\mathbb{N}\ m\urcorner + \\ {}_{\mathrm{ML}}\ulcorner mk\_\mathbb{N}\ n\urcorner\urcorner \end{array}$$

Uses For doing fast arithmetic proofs.

Errors

6085 ?0 is not of the form: $\ulcorner {}_{\mathrm{ML}}\ulcorner mk\_\mathbb{N}\ m\urcorner + {}_{\mathrm{ML}}\ulcorner mk\_\mathbb{N}\ n\urcorner\urcorner$

SML

```
val prim_rewrite_conv : CONV NET -> CANON -> (THM -> TERM * CONV) OPT ->
        (CONV -> CONV) -> EQN_CXT -> THM list -> CONV;
```

Description The primitive rewrite conversion.

Conversion

$$\frac{}{\Gamma \vdash t = t'} \quad \begin{array}{l} prim\_rewrite\_conv \\ (initial\_net\text{: } CONV\ NET) \\ (canon : CANON) \\ (eqm\_rule : (THM -> TERM * CONV)\ OPT) \\ (traverse : CONV -> CONV) \\ (with\_eqn\_cxt : EQN\_CXT) \\ (with\_thms : THM\ list)\ \ulcorner t \urcorner \end{array}$$

where ⌜t'⌝ is ⌜t⌝, rewritten according to the parameters of the conversion, and Γ are the assumptions required to allow the rewriting. The failure of the conversion constructed by *prim_rewrite_conv* will not be caught by *prim_rewrite_conv*.

The arguments have the following effects:

initial_net This is a pre-calculated conversion net, that will serve as the initial rewriting that may be done.

canon This canonicalisation function will be applied to all of the *with_thms* theorems, to produce a list of theorems to be rewritten with from these inputs. This will generally involve producing canonical or simplified forms of the original theorems.

The resulting theorems are intended to be simply universally quantified equations, and theorems which are not of this form are discarded. Rewriting attempts to instantiate some or all of the universally quantified variables, or any type variables (which do not appear in the assumptions), so as to to match the left-hand side of an equation to the term being rewritten. N.b. free variables are not instantiated. An equation whose left-hand side matches the term being rewritten in such a way that rewriting would not change the term is treated as if it did not match the term.

eqm_rule This equation matcher is mapped over the theorems resulting from the canonicalisation to convert them into an equation context. *thm_eqn_cxt* is used if *Nil* is supplied.

traverse This is a conversional, which defines the traversal of term t by the rewriting conversion derived from *prim_rewrite_conv*'s other arguments.

with_eqn_cxt This is additional equational context to be added directly into the rewriting conversion net.

with_thms This is an additional set of theorems to be processed by *canon* and the results used in added directly into the rewriting conversion net.

Uses This is the basis of the primary rewriting tools, by varying the first four parameters.

prim_rewrite_conv preprocesses its arguments in various ways. The preprocessing for an argument takes place as soon as that argument is supplied, so, for example, the overhead of preprocessing *with_eqn_cxt* need not be incurred in calls with the same *with_eqn_cxt* but different *with_thms*.

SML

val **prim_rewrite_rule** : $CONV\ NET$ −> $CANON$ −> $(THM$ −> $TERM * CONV)\ OPT$ −> $(CONV$ −> $CONV)$ −> EQN_CXT −> $THM\ list$ −> THM −> THM;

Description This is the inference rule based on *prim_rewrite_conv* (q.v.), with the same parameters as that function, except for the last argument:

Rule

$$\frac{\Gamma \vdash t}{\Gamma \cup \Gamma 1 \vdash t'}$$

prim_rewrite_rule
(*initial_net*: $CONV\ NET$)
(*canon* : $CANON$)
(*epp* : $(THM$ −> $TERM * CONV)\ OPT$)
(*traverse* : $CONV$ −> $CONV$)
(*with_eqn_cxt* : EQN_CXT)
(*with_thms* : $THM\ list$)

where ⌜t'⌝ is the result of rewriting ⌜t⌝ in the manner prescribed by the arguments, and $\Gamma 1$ are the assumptions required to allow this rewriting.

SML

val **prim_suc_conv** : $CONV$;

Description This conversion gives the definition schema for all natural number literals.

Rule

$$\frac{}{\vdash {}_{\mathrm{ML}}\ulcorner(mk\_\mathbb{N}(m{+}1))\urcorner = Suc\ {}_{\mathrm{ML}}\ulcorner mk\_\mathbb{N}\ m\urcorner}$$

prim_suc_conv
($mk\_\mathbb{N}$ $(m{+}1)$)

Rule

$$\frac{}{\vdash {}_{\mathrm{ML}}\ulcorner(mk\_\mathbb{N}\ 0)\urcorner = Zero}$$

prim_suc_conv
($mk\_\mathbb{N}$ 0)

Errors

3026 ?0 is not a numeric literal

See Also $mk\_\mathbb{N}$, *suc_conv*

SML

val **prove_asm_rule** : THM −> THM −> THM;

Description Eliminate an assumption with reference to a the assumption being a conclusion of a theorem.

Rule

$$\frac{\Gamma 1 \vdash t1;\ \Gamma 2,\ t1 \vdash t2}{\Gamma 1 \cup \Gamma 2 \vdash t2}$$

prove_asm_rule

This will in fact work even if the assumption is not present.

SML

|*val* **RANDS_C** : *CONV* $->$ *CONV*;

Description Apply a conversion to each of the arguments of a a function

Rule

$$\frac{}{\vdash f\ a\ ...\ z = f\ a'...z'} \quad \begin{array}{l} RANDS\_C \\ (c : CONV) \\ \ulcorner f\ a\ ...\ z \urcorner \end{array}$$

where $c\ a$ gives $\vdash a = a'$, etc. The function f may have no arguments in which case *refl_conv* f is returned.

Errors

|*7104* *Result of conversion, ?0, ill−formed*

Also as the failure of the conversion.

SML

|*val* **RAND_C** : *CONV* $->$ *CONV*;

Description Apply a conversion to the operand of a combination:

Rule

$$\frac{}{\vdash f\ a = f\ a'} \quad \begin{array}{l} RAND\_C \\ (c : CONV) \\ \ulcorner f\ a \urcorner \end{array}$$

where $c\ a$ gives $\vdash a = a'$.

Errors

|*3010* *?0 is not of form:* $\ulcorner$*t1 t2*$\urcorner$
|*7104* *Result of conversion, ?0, ill−formed*

Also as the failure of the conversion.

SML

|*val* **RATOR_C** : *CONV* $->$ *CONV*;

Description Apply a conversion to the operator of a combination:

Rule

$$\frac{}{\vdash f\ a = f'\ a} \quad \begin{array}{l} RATOR\_C \\ (c : CONV) \\ \ulcorner f\ a \urcorner \end{array}$$

where $c\ f$ gives '$\vdash f = f'$'.

Errors

|*3010* *?0 is not of form:* $\ulcorner$*t1 t2*$\urcorner$
|*7104* *Result of conversion, ?0, ill−formed*

Also as the failure of the conversion.

SML

|*val* **refl_conv** : *CONV*;

Description The reflexivity of equality implemented as a conversion.

Rule

$$\frac{}{\vdash t = t} \quad \begin{array}{l} refl\_conv \\ \ulcorner t \urcorner \end{array}$$

A primitive inference rule.

SML
```
val REPEAT_C1 : CONV -> CONV;
```

Description Repeatedly apply a conversion to a term, failing if not successfully applied at least once. To be more precise, the functionality is equivalent that of the following definition:

```
fun REPEAT_C1 (c:CONV) = (c THEN_TRY_C REPEAT_C1 c)
```

Errors As the error of the conversion if it cannot be applied at least once.

SML
```
val REPEAT_CAN : CANON -> CANON
```

Description *REPEAT_CAN* is a canonicalisation function combinator which repeatedly applies its argument until it fails:

```
REPEAT_CAN can thm =
	((can THEN_CAN REPEAT_CAN can) ORELSE_CAN id_can) thm
```

See Also *CANON*

SML
```
val REPEAT_C : CONV -> CONV;
```

Description Repeatedly apply a conversion to a term. To be more precise, the functionality is equivalent that of the following definition:

```
fun REPEAT_C (c:CONV) =
	(c THEN_C (REPEAT_C c)) ORELSE_C refl_conv
```

See Also *REPEAT_C1*

SML
```
val REPEAT_MAP_C : CONV -> CONV;
```

Description This traverses a term from its leaves to its root node. It will attempt the application of its conversion argument on each subterm encountered en route. If the conversion is successfully applied to a given sub-term, then the resulting sub-term from the conversion is re-traversed by the function. It traverses from left to right, though this should only matter for conversions that work by side-effect. It fails if the conversion is not applicable anywhere within the term, or if certain applications of the conversion have ill-formed results.

Errors
```
7005	Conversion fails on term and all its subterms
```

SML
```
val REWRITE_CAN : CANON -> CANON;
```

Description For rewriting, after all other canonicalisation we will usually wish to then universally quantify the resulting theorems in all free variables that are only in in the conclusion, other than those that were free anywhere in the original theorem, before any canonicalisation. A canonicalisation is transformed to work this way by *REWRITE_CAN*.

When evaluating proof contexts (see, e.g., *commit_pc*) the list of rewrite canonicalisations in the argument (see *get_rw_canons*), *arg*, will be converted to a single canonicalisation in the result by:

```
REWRITE_CAN
	(REPEAT_CAN(FIRST_CAN (arg @
		[⇔_t_rewrite_canon])));
```

SML
```
val rewrite_conv : THM list -> CONV;
val pure_rewrite_conv : THM list -> CONV;
val once_rewrite_conv : THM list -> CONV;
val pure_once_rewrite_conv : THM list -> CONV;
```

Description These are the standard rewriting conversions. They use the canonicalisation rule held by the proof context (see, e.g, *push_pc*) preprocess the theorem list. The context is accessed at the point when the rules are given a list of theorems.

If a conversion is "pure" then there is no default rewriting, otherwise the default rewriting conversion net held by the proof context will be used in addition to user supplied material.

If a conversion is "once" then rewriting will proceed from the root of the of the conclusion of the theorem to be rewritten, towards the leaves, and will not descend through any rewritten subterm, using *ONCE_MAP_WARN_C*. If not, rewriting will continue, moving from the root to the leaves, repeating if any rewriting is successful, until there is no rewriting redex anywhere within the rewritten conclusion, using *REWRITE_MAP_C*. This may cause non-terminating looping.

Errors
```
26001 no rewriting occurred
```

Also as error 26003 and warning 26002 of *REWRITE_MAP_C* (q.v.).

SML
```
val REWRITE_MAP_C : string -> CONV -> CONV;
```

Description This conversional is an equivalent to *TOP_MAP_C* (q.v.) except that it warns the user if it failed to recompose the theorems from the term it just traversed.

REWRITE_MAP_C conv tm traverses *tm* from its root node to its leaves. It will repeat the application of *conv*, until failure, on each subterm encountered en route. It then descends through the sub-term that results from the repeated application of *conv*. If the descent causes any change, on "coming back out" to the sub-term the conversional will attempt to reapply *conv*, and if successful will then (recursively) reapply *REWRITE_MAP_C conv* once more. If *conv* cannot be reapplied then the conversional continues to ascend back to the root.

It traverses from left to right, though this should only matter for conversions that work by side-effect. It fails if the conversion is applied nowhere within the term.

Errors
```
26001 no rewriting occurred
26003 no successful rewriting occurred, rewriting gave ill-formed results on some subterms
```

It issues the following warning message if at any point it fails to recompose the theorems from the subterm it just traversed, some successful rewriting occurs, and the flag "*illformed_rewrite_warning*" is true.

Errors
```
26002 rewriting gave ill-formed results on some subterms
```

Errors and warnings are from the area indicated by the string argument.

SML

```
val rewrite_rule : THM list -> THM -> THM;
val pure_rewrite_rule : THM list -> THM -> THM;
val once_rewrite_rule : THM list -> THM -> THM;
val pure_once_rewrite_rule : THM list -> THM -> THM;
val asm_rewrite_rule : THM list -> THM -> THM;
val pure_asm_rewrite_rule : THM list -> THM -> THM;
val once_asm_rewrite_rule : THM list -> THM -> THM;
val pure_once_asm_rewrite_rule : THM list -> THM -> THM;
```

Description These are the standard rewriting rules. They use the canonicalisation rule held by the proof context (see, e.g, *push_pc*) to preprocess the theorem list. The context is accessed at the point when the rules are given a list of theorems.

If a rule is "pure" then there is no default rewriting, otherwise the default rewriting conversion net held by the proof context will be used in addition to user supplied material.

If a rule is "once" then rewriting will proceed from the root of the of the conclusion of the theorem to be rewritten, towards the leaves, and will not descend through any rewritten subterm, using *ONCE_MAP_WARN_C*. If not, rewriting will continue, moving from the root to the leaves, repeating if any rewriting is successful, until there is no rewriting redex anywhere within the rewritten conclusion, using *REWRITE_MAP_C*. This may cause non-terminating looping.

If a rule is "asm" then the theorems rewritten with will include the canonicalised *asm_rule*d assumptions of the theorem being rewritten.

See Also *prim_rewrite_rule*

Errors

```
26001 no rewriting occurred
```

Also as error 26003 and warning 26002 of *REWRITE_MAP_C*(q.v.).

SML

```
val RIGHT_C : CONV -> CONV;
```

Description Apply a conversion to the second operand of a binary operator:

Rule

$$\frac{}{\vdash f\ a\ b = f\ a\ b'} \quad \begin{array}{l} RIGHT\_C \\ (c : CONV) \\ \ulcorner f\ a\ b \urcorner \end{array}$$

where $c\ b$ gives '$\vdash b = b'$'. f may itself be a function application.

Errors

```
3013 ?0 is not of form: ⌜f a b⌝
7104 Result of conversion, ?0, ill-formed
```

Also as the failure of the conversion.

SML

val **SIMPLE_BINDER_C** : $CONV$ $->$ $CONV$;

Description Apply a conversion to the body of a simple binder term:

Rule

$$\frac{}{\vdash (B\ x \bullet p[x]) = (B\ x \bullet p'[x])} \quad \begin{array}{l} SIMPLE\_BINDER\_C \\ (c : CONV) \\ \ulcorner B\ x \bullet p \urcorner \end{array}$$

where $c\ p[x]$ gives '$\vdash p[x] = p'[x]$', and B is a binder.

Errors

7059	*?0 is not of the form:* $\ulcorner B\ x \bullet p[x] \urcorner$ *where* $\ulcorner B \urcorner$ *is a binder and* $\ulcorner x \urcorner$ *a variable*
7104	*Result of conversion, ?0, ill−formed*

Also as the failure of the conversion.

SML

val **simple_eq_match_conv** : THM $->$ $CONV$;

Description This matches the LHS of an equational theorem to a term, instantiating the RHS accordingly. In fact the equation may be partially or fully universally quantified (simple quantification only), without affecting the result of the conversion.

Conversion

$$\frac{}{\Gamma' \vdash t = v'} \quad \begin{array}{l} simple\_eq\_match\_conv \\ (\Gamma \vdash \forall \ldots \bullet u = v) \\ \ulcorner t \urcorner \end{array}$$

where v' is the result of applying to v the instantiation rules required to match u to t (including both term and type instantiation). If there are free variables on the RHS of the supplied equational theorem (when stripped of all universal quantification) they will be renamed as necessary to avoid identification with any variables in t.

Errors

7044	*Cannot match ?0 and ?1*

SML

```
val simple_eq_match_conv1 : THM -> CONV ;
```

Description This matches the LHS of an universally quantified (simple quantifiers only) equational theorem to a term, instantiating the RHS accordingly. The conversion will only instantiate its universal quantifications, and type variables not present in the assumptions, and not its free term variables.

Conversion

$$\frac{}{\Gamma \vdash t = v[t1,...,tn]} \quad \begin{array}{l} simple\_eq\_match\_conv1 \\ (\Gamma \vdash \forall\ x1\ ...\ xn \bullet u[x1,...,xn] = \\ \quad v[x1,...,xn]) \\ \ulcorner t \urcorner \end{array}$$

where ⌜u[t1,...,tn]⌝ is α-convertible to ⌜t⌝. If there are free variables on the RHS of the supplied equational theorem(when stripped of all universal quantification) they will be renamed as necessary to avoid identification with any variables in t.

This conversion may be partially evaluated with only its theorem argument.

Uses In producing a limited rewriting facility, that only instantiates explicitly identified variables.

Errors

7095	?0 is not of the form ‘Γ ⊢ ∀ x1 ... xn • u = v‘ where ⌜xi⌝ are variables
7076	Could not match term ?0 to LHS of theorem ?1

SML

```
val simple_ho_eq_match_conv : THM -> CONV
```

Description This conversion is like *simple_eq_match_conv* but uses higher-order matching. It uses *ho_match* (q.v.) to match the LHS of an equational theorem to a term t. It then instantiates the theorem (including both term and type instantiation) and carries out any $\beta\eta$-reductions required to give a theorem of the form $t = v'$. The equation may be partially or fully universally quantified (simple quantification only, not quantification over pairs).

Conversion

$$\frac{}{\Gamma' \vdash t = v'} \quad \begin{array}{l} simple\_ho\_eq\_match\_conv \\ (\Gamma \vdash \forall\ ... \bullet u = v) \\ \ulcorner t \urcorner \end{array}$$

where v' is the result of applying to v the instantiations required to match u to t (including term and type instantiation). If there are free variables on the RHS of the supplied equational theorem (when stripped of all universal quantification) they will be renamed as necessary to avoid identification with any variables in t.

Errors

7095	?0 is not of the form ‘Γ ⊢ ∀ x1 ... xn • u = v‘ where ⌜xi⌝ are variables
7076	Could not match term ?0 to LHS of theorem ?1

SML

```
val simple_ho_eq_match_conv1 : THM -> CONV
```

Description This conversion is like *simple_eq_match_conv1* but uses higher-order matching. It uses *ho_match* (q.v.) to match the LHS of an equational theorem to a term t. The equation may be partially or fully universally quantified (simple quantification only, not quantification over pairs). It instantiates the theorem (including both term and type instantiation) and carries out any $\beta\eta$-reductions required to give a theorem of the form $t = v'$. Only type variables that do not appear in the assumptions of the theorem and universally quantified term variables will be instantiated.

Conversion

$$\frac{}{\Gamma \vdash t = v'} \quad \begin{array}{l} simple\_ho\_eq\_match\_conv1 \\ (\Gamma \vdash \forall \ldots \bullet u = v) \\ \ulcorner t \urcorner \end{array}$$

where v' is the result of applying to v the instantiation rules required to match u to t (including term and type instantiation). If there are free variables on the RHS of the supplied equational theorem (when stripped of all universal quantification) they will be renamed as necessary to avoid identification with any variables in t.

Errors

7095	?0 is not of the form '$\Gamma \vdash \forall$ x1 ... xn $\bullet$ u = v' where ⌜xi⌝ are variables
7076	Could not match term ?0 to LHS of theorem ?1

SML

```
val simple_⇔_match_mp_rule : THM -> THM -> THM;
```

Description A matching Modus Ponens for $\Leftrightarrow$.

Rule

$$\frac{\Gamma 1 \vdash \forall\ x1\ \ldots \bullet t1 \Leftrightarrow t2;\ \Gamma 2 \vdash t1'}{\Gamma 1' \cup \Gamma 2 \vdash t2'} \quad simple\_\Leftrightarrow\_match\_mp\_rule$$

where $t1'$ is an instance of $t1$ under type instantiation and substitution for the x_i and the free variables of the first theorem, and where $t2'$ is the corresponding instance of $t2$. No type instantiation or substitution will occur in the assumptions of either theorem.

See Also *⇒_elim* (Modus Ponens on ⇒), *simple_⇔_match_mp_rule*

Errors

7044	Cannot match ?0 and ?1
7046	?0 is not of the form '$\Gamma \vdash \forall$ x1 ... xn $\bullet$ u $\Leftrightarrow$ v'

SML

```
val simple_⇔_match_mp_rule1 : THM -> THM -> THM;
```

Description A matching Modus Ponens for $\Leftrightarrow$ that doesn't affect assumption lists.

Rule

$$\frac{\Gamma 1 \vdash \forall\ x1\ \ldots \bullet t1 \Leftrightarrow t2;\ \Gamma 2 \vdash t1'}{\Gamma 1 \cup \Gamma 2 \vdash t2'} \quad simple\_\Leftrightarrow\_match\_mp\_rule1$$

where $t1'$ is an instance of $t1$ under type instantiation and substitution for the x_i (but not free variables), and where $t2'$ is the corresponding instance of $t2$. Types in the assumptions of the theorems will not be instantiated.

See Also *⇒_elim* (Modus Ponens on ⇒), *simple_⇔_match_mp_rule1*

Errors

7044	Cannot match ?0 and ?1
7046	?0 is not of the form '$\Gamma \vdash \forall$ x1 ... xn $\bullet$ u $\Leftrightarrow$ v'

SML
```
val simple_⇒_match_mp_rule : THM -> THM -> THM ;
```

Description A matching Modus Ponens rule for an implicative theorem.

Rule

$$\frac{\Gamma 1 \vdash \forall\ x1\ ...\ \bullet\ t1 \Rightarrow t2;\ \Gamma 2 \vdash t1'}{\Gamma 1 \cup \Gamma 2 \vdash t2'} \qquad simple\_\Rightarrow\_match\_mp\_rule$$

where $t1'$ is an instance of $t1$ under type instantiation and substitution for the x_i and the free variables of the first theorem, and where $t2'$ is the corresponding instance of $t2$. No type instantiation or substitution will occur in the assumptions of either theorem.

See Also *simple_⇒_match_mp_rule1*, *simple_⇒_match_mp_rule2*

Errors
```
7044   Cannot match ?0 and ?1
7045   ?0 is not of the form ‘Γ ⊢ ∀ x1 ... xn • u ⇒ v‘
```

SML
```
val simple_⇒_match_mp_rule1 : THM -> THM -> THM ;
val simple_⇒_match_mp_rule2 : THM -> THM -> THM ;
```

Description Two variants on a matching Modus Ponens rule for an implicative theorem.

Rule

$$\frac{\Gamma 1 \vdash \forall\ x1\ ...\ \bullet\ t1 \Rightarrow t2;\ \Gamma 2 \vdash t1'}{\Gamma 1 \cup \Gamma 2 \vdash t2'} \qquad simple\_\Rightarrow\_match\_mp\_rule1$$

where $t1'$ is an instance of $t1$ under type instantiation and substitution for the x_i (but not free variables), and where $t2'$ is the corresponding instance of $t2$.

simple_⇒_match_mp_rule2 is just like *simple_⇒_match_mp_rule1* except that the instantiations and substitutions returned by *term_match* are extended to replace type variables that do not occur in $t1$ or in $\Gamma 1$ and x_i that do not occur free in $t1$ by fresh variables to avoid clashes with each other and with the type variables and free variables of $\Gamma 1$ and $\Gamma 2$.

Types in the assumptions of the theorems will not be instantiated.

See Also *simple_⇒_match_mp_rule*

Errors
```
7044   Cannot match ?0 and ?1
7045   ?0 is not of the form ‘Γ ⊢ ∀ x1 ... xn • u ⇒ v‘
```

SML
```
val simple_∀_elim : TERM -> THM -> THM;
```

Description Instantiate a universally quantified variable to a given value.

Rule

$$\frac{\Gamma \vdash \forall\ x \bullet\ t2[x]}{\Gamma \vdash t2'[t1]} \qquad \begin{array}{l} simple\_\forall\_elim \\ \ulcorner t1 \urcorner \end{array}$$

where $t2'$ is renamed from $t2$ to prevent bound variable capture, and x is a variable.

Errors
```
3012   ?0 and ?1 do not have the same types
7039   ?0 is not of the form: ‘Γ ⊢ ∀ x • t‘ where ⌜x⌝ is a variable
```

SML

```
val simple_∀_intro : TERM -> THM -> THM;
```

Description Introduce a simple universally quantified theorem.

Rule

$$\frac{\Gamma \vdash t}{\Gamma \vdash \forall\ x \bullet t} \qquad \begin{array}{l} \mathit{simple\_\forall\_intro} \\ \ulcorner x \urcorner \end{array}$$

A built-in inference rule.

See Also *∀_intro*

Errors

3007 ?0 is not a term variable
6005 ?0 occurs free in assumption list

SML

```
val simple_∀_∃_conv: CONV;
```

Description Swap the order of a simple ∀ and ∃:

Conversion

$$\frac{}{\vdash (\forall\ x \bullet \exists\ y \bullet\ P[x,y]) \Leftrightarrow (\exists\ y' \bullet \forall\ x \bullet\ P[x,\ y'\ x])} \qquad \begin{array}{l} \mathit{simple\_\forall\_\exists\_conv} \\ \ulcorner \forall\ x \bullet \exists\ y \bullet\ P[x,y] \urcorner \end{array}$$

where y' is renamed to distinguish it from y (for the types differ) and every other term variable in the argument.

Errors

27031 ?0 is not of the form: $\ulcorner \forall\ x \bullet \exists\ y \bullet\ P[x,y] \urcorner$

SML

```
val simple_∃_elim : TERM -> THM -> THM -> THM ;
```

Description Eliminate an existential quantifier.

Rule

$$\frac{\Gamma 1 \vdash \exists\ x \bullet\ t1[x];\ \Gamma 2,\ t1[y] \vdash t2}{\Gamma 1 \cup \Gamma 2 \vdash t2} \qquad \begin{array}{l} \mathit{simple\_\exists\_elim} \\ \ulcorner y \urcorner \end{array}$$

where y must be variable which is not present elsewhere in the second theorem, nor in the conclusion of the first. $t1[y]$ need not actually be present in the assumptions of the second theorem.

Errors

3007 ?0 is not a term variable
7014 ?0 has the wrong type
7109 ?0 is not of the form '$\Gamma \vdash \exists\ x \bullet\ t[x]$*'*
7120 ?0 occurs free in conclusion of ?1
7121 ?0 occurs free in hypotheses of ?1 other than ?2

SML
```
val simple_∃_intro : TERM -> THM -> THM ;
```

Description Introduce an existential quantifier by reference to a witness.

Rule

$$\frac{\Gamma \vdash t1[t2]}{\Gamma \vdash \exists\ x \bullet\ t1[x]} \qquad \begin{array}{l} simple\_\exists\_intro \\ ⌜\exists\ x \bullet\ t1[x]⌝ \end{array}$$

where ⌜x⌝ is a variable.

Errors

3034 ?0 is not of form: ⌜∃ *var* • *body*⌝
7047 ?0 cannot be matched to conclusion of theorem ?1

SML
```
val simple_∃_∀_conv: CONV;
```

Description Swap the order of a simple ∃ and ∀:

Conversion

$$\frac{}{\begin{array}{c}\vdash (\exists\ x\bullet\ \forall\ y\bullet\ P[x,y]) \Leftrightarrow \\ (\forall\ y'\bullet\ \exists\ x\bullet\ P[x,\ y'\ x])\end{array}} \qquad \begin{array}{l} simple\_\exists\_\forall\_conv \\ ⌜\exists\ x\bullet\ \forall\ y\bullet\ P[x,y]⌝ \end{array}$$

where y' is renamed to distinguish it from y (for the types differ) and every other term variable in the argument.

Errors

27032 ?0 is not of the form: ⌜∃ *x*• ∀ *y*• *P*[*x*,*y*]⌝

SML
```
val simple_∃_∀_conv1: CONV;
```

Description Swap the order of a simple ∃ and ∀, where the first variable is always applied to the second:

Conversion

$$\frac{}{\begin{array}{c}(\exists\ f\bullet\ \forall\ x\bullet\ P[f\ x,\ x]) \Leftrightarrow \\ (\forall\ x\bullet\ \exists\ f'\bullet\ P[f',\ x])\end{array}} \qquad \begin{array}{l} simple\_\exists\_\forall\_conv1 \\ ⌜\exists\ f\bullet\ \forall\ x\bullet\ P[f\ x,\ x]⌝ \end{array}$$

where f' is renamed to distinguish it from f (for the types differ) and every other term variable in the argument.

Errors

27033 ?0 is not of the form: ⌜∃ *f*• ∀ *x*• *P*[*f x*,*x*]⌝

SML
```
val simple_∃_ϵ_conv : CONV;
```

Description Give that ϵ of a predicate satisfies the predicate by reference to an ∃ construct.

Rule

$$\frac{}{\Gamma \vdash (\exists\ x \bullet\ p[x]) \Leftrightarrow p[\epsilon\ x \bullet\ p\ [x]]} \qquad \begin{array}{l} simple\_\exists\_\epsilon\_conv \\ ⌜\exists\ x \bullet\ p[x]⌝ \end{array}$$

See Also *∃_ϵ_rule*

Errors

3034 ?0 is not of form: ⌜∃ *var* • *body*⌝

SML

val **simple_$\exists$_ϵ_rule** : *THM* -> *THM*;

Description Give that ϵ of a predicate satisfies the predicate by reference to an $\exists$ construct. It can properly handle paired existence.

Rule

$$\frac{\Gamma \vdash \exists\ x \bullet p[x]}{\Gamma \vdash p[\epsilon\ x \bullet p\ x]} \qquad simple\_\exists\_\epsilon\_rule$$

See Also $\exists\_\epsilon\_conv$

Errors

7092 *?0 is not of the form:* $`\Gamma \vdash \exists\ x \bullet p[x]`$

SML

val **simple_$\exists_1$_elim** : *THM* -> *THM*;

Description Express a $\exists_1$ in terms of $\exists$ and a uniqueness property.

Rule

$$\frac{\Gamma \vdash \exists_1\ x \bullet P[x]}{\Gamma \vdash \exists\ x \bullet P[x] \land \forall\ y \bullet P[y] \Rightarrow y = x} \qquad simple\_\exists_1\_elim$$

Errors

7015 *?0 is not of the form:* $`\Gamma \vdash \exists_1\ x \bullet P[x]`$

SML

val **simple_$\exists_1$_intro** : *THM* -> *THM* -> *THM*;

Description Introduce $\exists_1$ by reference to a witness, and a uniqueness theorem.

Rule

$$\frac{\begin{array}{c}\Gamma 1 \vdash P'[t'] \\ \Gamma 2 \vdash \forall\ x \bullet P[x] \Rightarrow x = t\end{array}}{\Gamma 1 \cup \Gamma 2 \vdash \exists_1\ x \bullet P[x]} \qquad simple\_\exists_1\_intro$$

Where P' is α-convertible to P, and t' is α-convertible to t. Notice that for the resulting theorem we take the bound variable name, x, and the form of the predicate, P, from the second theorem.

Errors

7066 *?0 not of the form:* $`\Gamma \vdash \forall\ x \bullet P[x] \Rightarrow x = t`$

7067 *?0 and ?1 are not of the form:* $`\Gamma 1 \vdash Pa[ta]`$ *and* $`\Gamma 2 \vdash \forall\ x \bullet P[x] \Rightarrow x = t`$ *where* $\ulcorner Pa \urcorner$ *and* $\ulcorner P \urcorner$, $\ulcorner ta \urcorner$ *and* $\ulcorner t \urcorner$ *are* α*−convertible*

SML

val **simple_α_conv** : *string* -> *CONV*;

Description Rename a bound variable name, as a conversion. This only works with simple abstractions.

Rule

$$\frac{}{\vdash (\lambda\ x \bullet t[x]) = (\lambda\ v \bullet t[v])} \qquad \begin{array}{l} simple\_\alpha\_conv \\ (v : string) \\ \ulcorner \lambda\ x \bullet t[x] \urcorner \end{array}$$

Errors

3011 *?0 is not of form:* $\ulcorner \lambda\ var \bullet t \urcorner$

7035 *Cannot rename bound variable ?0 to ?1 as this would cause variable capture*

SML

|*val* **simple_β_conv** : *CONV*;

Description Apply a β-reduction to a simple abstraction.

Conversion

$$\frac{}{\vdash (\lambda\ x \bullet t1[x])\ t2 = t1[t2]} \qquad \begin{array}{l} simple\_\beta\_conv \\ \ulcorner(\lambda\ x \bullet t1[x])\ t2\urcorner \end{array}$$

A primitive inference rule.

See Also *β_conv*

Errors

|*6012 ?0 is not of the form:* $\ulcorner(\lambda\ x \bullet t1[x])t2\urcorner$ *where* $\ulcorner x\urcorner$ *is a variable*

SML

|*val* **simple_β_η_conv** : *TERM* −> *CONV*;

Description If t is any term, *simple_β_η_conv t* is a conversion which will prove all theorems of the form $\vdash t = s$ where t and s are simply $\alpha\beta\eta$-equivalent, i.e., can be reduced to α-equivalent normal forms by β- and η-reduction involving only simple (rather than paired) λ-abstractions.

Errors

|*7131 ?0 and ?1 are not simply αβη−equivalent*

SML

|*val* **simple_β_η_norm_conv** : *CONV*;

Description This conversion eliminates all simple β- and η-redexes from a term giving the $\beta\eta$-normal form. It does not eliminate β- and η-redexes involving abstraction over pairs. It fails if the term is already in normal form.

Errors

|*7130 ?0 contains no simple β− or η−redexes*

SML

|*val* **simple_ϵ_elim_rule** : *TERM* −> *THM* −> *THM* −> *THM*;

Description Given that ϵ of a predicate satisfies that predicate, then in a different theorem we may eliminate an assumption that claims an otherwise unused variable satisfies the predicate.

Rule

$$\frac{\begin{array}{c} \Gamma 1 \vdash t'\ (\$\epsilon\ t''); \\ \Gamma 2,\ t\ x \vdash s \end{array}}{\Gamma 1 \cup \Gamma 2 \vdash s} \qquad \begin{array}{l} simple\_\epsilon\_elim\_rule \\ \ulcorner x\urcorner \end{array}$$

where t, t' and t'' are α-convertible, and x is a free variable whose only free occurrence in the second theorem is the one shown and which does not appear free in the conclusion of the first theorem. In fact, $(\$\epsilon\ t'')$ here can be any term, it is not constrained to be an application of the choice function.

Errors

|*3007 ?0 is not a term variable*
|*7019 ?0 is not of the form: ‘Γ ⊢ t1($ϵ t1)‘*
|*7054 ?0 is not of same type as choice sub−term of first theorem*
|*7108 Arguments not of the form* $\ulcorner?0\urcorner$*, ‘Γ1 ⊢ t ($ϵ t)‘ and ‘Γ2, (t ?0) ⊢ s‘*
|*7120 ?0 occurs free in conclusion of ?1*
|*7121 ?0 occurs free in hypotheses of ?1 other than ?2*
|*7122 ?0 occurs free in operator of the conclusion of ?1*

SML

```
val SIMPLE_λ_C : CONV -> CONV;
```

Description Apply a conversion to the body of a simple abstraction:

Rule

$$\frac{}{\vdash (\lambda\ x \bullet p[x]) = (\lambda\ x \bullet p'[x])} \quad \begin{array}{l} SIMPLE\_\lambda\_C \\ (c : CONV) \\ \ulcorner \lambda\ x \bullet p \urcorner \end{array}$$

where $c\ p[x]$ gives '$\vdash p[x] = p'[x]$'.

See Also *SIMPLE_BINDER_C*

Errors

3011	*?0 is not of form:* $\ulcorner \lambda\ var \bullet t \urcorner$
7104	*Result of conversion, ?0, ill−formed*

Also as the failure of the conversion.

SML

```
val simple_λ_eq_rule : TERM -> THM -> THM;
```

Description Given an equational theorem, return the equation formed by abstracting the term argument (which must be a variable) from both sides.

Rule

$$\frac{\Gamma \vdash t1[x] = t2[x]}{\Gamma \vdash (\lambda\ x \bullet t1[x]) = (\lambda\ x \bullet t2[x])} \quad \begin{array}{l} simple\_\lambda\_eq\_rule \\ \ulcorner x \urcorner \end{array}$$

A primitive inference rule.

See Also *λ_eq_rule*

Errors

3007	*?0 is not a term variable*
6005	*?0 occurs free in assumption list*
6020	*?0 is not of the form:* '$\Gamma \vdash t1 = t2$'

SML
|*val* **string_conv** : *CONV*;

Description This function defines the constants with names starting with ", and type *CHAR LIST* (an abbreviation of *CHAR LIST*). A string literal constant is indicated by the constant name starting with a double quote("), as well as being of type *CHAR LIST*. This is equivalent to a list of character literal constants, one for each but the first (") character of the string constant's name. This conversion defines this relationship, by returning the head and unexploded tail of the list of characters. A character literal is indicated by the constant's name starting with single backquote (‘), as well as being of type *CHAR*.

Rule

$$\frac{}{\vdash\ {}_{\mathrm{ML}}\ulcorner(mk\_string("c..."))\urcorner = Cons\ {}_{\mathrm{ML}}\ulcorner(mk\_char("c"))\urcorner\ {}_{\mathrm{ML}}\ulcorner(mk\_string("..."))\urcorner} \quad \begin{array}{l} string\_conv \\ (mk\_string\ ("c...")) \end{array}$$

Or:

Rule

$$\frac{}{\vdash\ {}_{\mathrm{ML}}\ulcorner(mk\_string(""))\urcorner = Nil} \quad \begin{array}{l} string\_conv \\ (mk\_string\ "") \end{array}$$

A primitive inference rule(axiom schemata).

See Also *mk_string*

Errors
|*3025 ?0 is not a string literal*

SML
|*val* **strip_∧_rule** : *THM -> THM list*;

Description Break a theorem into conjuncts as far as possible.

Rule

$$\frac{\Gamma \vdash t}{[\Gamma \vdash t1,\ ...,\ \Gamma \vdash tn]} \quad strip\_\wedge\_rule$$

where t can be formed from the t_i by *∧_intro* alone, with no duplication, exception or reordering.

Example
|*strip_∧_rule* ‘⊢ (a ∧ b) ∧ (a ∧ c ∧ d)‘
| =
|[‘⊢ a‘, ‘⊢ b‘, ‘⊢ a‘, ‘⊢ c‘, ‘⊢ d‘]

SML
|*val* **strip_⇒_rule** : *THM -> THM*;

Description Repeatedly apply *undisch_rule*:

Rule

$$\frac{\Gamma \vdash t1 \Rightarrow ... \Rightarrow tn \Rightarrow t}{\Gamma \cup \{t1,\ ...,\ tn\} \vdash t} \quad strip\_\Rightarrow\_rule$$

SML

val **subst_conv** : $(\mathit{THM} * \mathit{TERM})\ \mathit{list} \to \mathit{TERM} \to \mathit{CONV}$;

Description Substitution of equational theorems according to a template.

Conversion

$$\frac{}{\Gamma 1 \cup ...\ \Gamma n \vdash t[...,ti,...] = t[...,ti',...]} \quad \begin{array}{l} \mathit{subst\_conv} \\ [...,(`\Gamma i \vdash ti{=}ti'`,\ulcorner xi\urcorner),\ ...] \\ \ulcorner t[...,xi,...]\urcorner \\ \ulcorner t[...,ti,...]\urcorner \end{array}$$

subst_conv $[(thm\_1, x\_1), \ldots, (thm\_n, x\_n)]$ *template term* returns a theorem in which *template* determines where in *term* the *thm_i* are substituted, when forming the RHS of the equation. The *x_i* must be variables. The template is of the form $t[x\_1, \ldots, x\_n]$, and wherever the *x_i* are free in *template* their associated equational theorem, *thm_i*, is substituted into *thm*. The rule will rename as necessary to avoid bound variable capture. The assumption list of the resulting theorem will be the union of all substitution theorems, regardless of use.

The RHS of the resulting theorem will take its bound variable names from *template*, not *term*, as shown in the following example. This provides an α-conversion facility.

This function may be partially evaluated with only one argument.

Example

```
subst_conv [(`⊢ p = q`, ⌜x1⌝), (`⊢ r = s`, ⌜x2⌝)]
        (⌜∀ y • f x1 r y + g x2 p = h y⌝)
        (⌜∀ x • f p r x + g r p = h x⌝)
        =
        `⊢ (∀ x • f p r x + g r p = h x) ⇔
            ∀ y • f q r y + g s p = h y`
```

See Also *subst_rule*

Errors

3007	*?0 is not a term variable*
3012	*?0 and ?1 do not have the same types*
6001	*?0 does not substitute to conclusion of theorem ?1*
6002	*Substitution theorem ?0 is not of the form: `Γ ⊢ t1 = t2`*
6029	*Substitution list contains entry (?0,?1) where the type of the variable differs from the type of the LHS of the theorem*

SML

val **subst_rule** : $(THM * TERM)$ *list* −> *TERM* −> *THM* −> *THM*;

Description Substitution of equational theorems according to a template.

Rule

$$\frac{\begin{array}{c}[\Gamma 1 \vdash t1{=}t1',\ ...\ ,\ \Gamma n \vdash tn{=}tn'] \\ \Gamma \vdash t[t1,...tn]\end{array}}{\Gamma 1 \cup ...\ \Gamma n \cup \Gamma \vdash t[t1',...tn']} \quad subst\_rule$$

subst_rule $[(thm\_1, x\_1), \ldots, (thm\_n, x\_n)]$ *template thm* returns a theorem in which *template* determines where in *thm* the *thm_i* are substituted. The *x_i* must be variables. The template is of the form $t[x\_1, \ldots, x\_n]$, and wherever the *x_i* are free in *template* their associated equational theorem, *thm_i*, is substituted into *thm*. The rule will rename as necessary to avoid bound variable capture. The assumption list of the resulting theorem will be the union of all substitution theorems, regardless of use.

The conclusion of the resulting theorem will take its bound variable names from *template*, not *thm*, as shown in the following example. This provides an α-conversion facility.

The function may be usefully partially evaluated with one or two arguments.

A primitive inference rule.

Example

```
subst_rule [(‘⊢ p = q‘, ⌜x1⌝), (‘⊢ r = s‘, ⌜x2⌝)]
        (⌜∀ y • f x1 r y + g x2 p = h y⌝)
        (‘⊢ ∀ x • f p r x + g r p = h x‘)
        =
        ‘⊢ ∀ y • f q r y + g s p = h y‘
```

See Also *subst_conv*

Errors

3007	*?0 is not a term variable*
6001	*?0 does not substitute to conclusion of theorem ?1*
6002	*Substitution theorem ?0 is not of the form: ‘Γ ⊢ t1 = t2‘*
6029	*Substitution list contains entry (?0,?1) where the type of the variable differs from the type of the LHS of the theorem*

SML

val **SUB_C1** : $CONV$ −> $CONV$;

Description Apply a conversion to each of the constituents of a term, failing if the term cannot be broken up, or the conversion fails on all constituents (if only one of the two constituents of a *mk_app* have failures, then the offending term will be *refl_conv*ed instead). Thus:

$$SUB\_C1\ cnv\ var = fail\_conv\ var$$

$$SUB\_C1\ cnv\ const = fail\_conv\ const$$

$$\begin{aligned} &SUB\_C1\ cnv\ (f\ x) = \Gamma \vdash f\ x = f'\ x' \\ &\quad where\ cnv\ f = \Gamma 1 \vdash f = f' \\ &\quad and\ \ \ cnv\ x = \Gamma 2 \vdash x = x' \\ &\quad and\ \Gamma = \Gamma 1 \cup \Gamma 2 \end{aligned}$$

$$\begin{aligned} &SUB\_C1\ cnv\ (\lambda\ x \bullet t) = \Gamma \vdash (\lambda\ x \bullet t) = (\lambda\ x \bullet t') \\ &\quad where\ cnv\ t = \Gamma \vdash t = t' \end{aligned}$$

Errors

7104	*Result of conversion, ?0, ill−formed*
7105	*?0 has no constituents*

There may be failure messages from the conversions.

SML

val **SUB_C** : $CONV$ −> $CONV$;

Description Apply a conversion to each of the constituents of a term, however that term might be constructed, and recombine the results. Thus:

$$SUB\_C\ cnv\ var = refl\_conv\ var$$

$$SUB\_C\ cnv\ const = refl\_conv\ const$$

$$\begin{aligned} &SUB\_C\ cnv\ (f\ x) = \Gamma \vdash f\ x = f'\ x' \\ &\quad where\ cnv\ f = \Gamma 1 \vdash f = f' \\ &\quad and\ \ \ cnv\ x = \Gamma 2 \vdash x = x' \\ &\quad and\ \Gamma = \Gamma 1 \cup \Gamma 2 \end{aligned}$$

$$\begin{aligned} &SUB\_C\ cnv\ (\lambda\ x \bullet t) = \Gamma \vdash (\lambda\ x \bullet t) = (\lambda\ x \bullet t') \\ &\quad where\ cnv\ t = \Gamma \vdash t = t' \end{aligned}$$

See Also *SUB_C1*

SML

val **suc_conv** : *CONV*;

Description This conversion gives the definition schema for non-zero natural number literals.

Rule

$$\frac{}{\vdash \ _{\mathrm{ML}}\ulcorner(mk\_\mathbb{N}(m+1))\urcorner = Suc\ _{\mathrm{ML}}\ulcorner mk\_\mathbb{N}\ m\urcorner} \quad \begin{array}{l} suc\_conv \\ (mk\_\mathbb{N}\ (m+1)) \end{array}$$

The conversion fails if given *0*.

Errors

3026 ?0 is not a numeric literal
7100 ?0 must be numeric literal > 0

See Also *mk_*ℕ, *prim_suc_conv*

SML

val **THEN_CAN** : (*CANON* ∗ *CANON*) −> *CANON*

Description *THEN_CAN* is a canonicalisation function combinator written as an infix operator. (*can1 THEN_CAN can2*)*thm* is the result of applying *can2* to each of the theorems in the list *can1 thm* and then flattening the resulting list of lists.

See Also *CANON*

SML

val **THEN_C** : (*CONV* ∗ *CONV*) −> *CONV*;

Description Combine the effect of two successful conversions.

Rule

$$\frac{}{\Gamma \vdash t = t'''} \quad \begin{array}{l} (c1\text{: } CONV)\ THEN\_C\ (c2\text{: } CONV) \\ \ulcorner t\urcorner \end{array}$$

where *c1* t returns '$\Gamma 1 \vdash t = t'$', *c2* t' returns '$\Gamma 2 \vdash t'' = t'''$', t' and t'' are α-convertible and Γ equals $\Gamma 1 \cup \Gamma 2$.

See Also *EVERY_C* (the iterated version of this function), as well as *THEN_TRY_C*, *AND-_OR_C*, and *ORELSE_C*

Errors

7101 Result of first conversion, ?0, not an equational theorem
7102 LHS (if any) of result of second conversion, ?0, not
α−convertible to RHS of first, ?1

Errors If any, as the failures of *c1* and *c2* applied to t and t' respectively.

SML

val **THEN_LIST_CAN** : (*CANON* ∗ *CANON list*) −> *CANON*

Description *THEN_LIST_CAN* is a canonicalisation function combinator written as an infix operator. (*can1 THEN_LIST_CAN cans*)*thm* is the result of applying each element of the list *cans* to the corresponding element of the list *can1 thm* and then flattening the resulting list of lists.

See Also *CANON*

Errors

26204 wrong number of canonicalisation functions in the list

SML

```
val THEN_TRY_C : (CONV * CONV) -> CONV;
```

Description Combine the effect of two conversions, ignoring the failure of the second if necessary. That is, if the first conversion results in an equational theorem whose RHS can have the second conversion applied, and the two resulting theorems composed, then that composition; otherwise the result of the first conversion alone is returned.

See Also *THEN_C*, *AND_OR_C*, *ORELSE_C*

Errors As the failure of *c1*.

SML

```
val TOP_MAP_C : CONV -> CONV;
```

Description *TOP_MAP_C conv tm* traverses *tm* from its root node to its leaves. It will repeat the application of *conv*, until failure, on each subterm encountered en route. It then descends through the sub-term that results from the repeated application of the conversion. If the descent causes any change, on "coming back out" to the sub-term the conversional will attempt to reapply *conv*, and if successful will then (recursively) reapply *TOP_MAP_C conv* once more. If *conv* cannot be reapplied then the conversional continues to ascend back to the root.

It traverses from left to right, though this should only matter for conversions that work by side-effect. It fails if the conversion is applied nowhere within the term.

Errors

7005	*Conversion fails on term and all its subterms*

SML

```
val TRY_C : CONV -> CONV;
```

Description Attempt to apply a conversion, and if it fails, apply *refl_conv*.

SML

```
val t_thm : THM;
```

Description "True" is true.

Theorem

$$\frac{}{\vdash T} \qquad t\_thm$$

SML

```
val undisch_rule : THM -> THM ;
```

Description Undischarge the antecedent of an implicative theorem into the assumption list.

Rule

$$\frac{\Gamma \vdash a \Rightarrow b}{\Gamma \cup \{a\} \vdash b} \qquad undisch\_rule$$

Errors

7011	*?0 is not of the form*: '$\Gamma \vdash a \Rightarrow b$'

SML
```
val varstruct_variant : TERM list -> TERM -> TERM;
```

Description *varstruct_variant avoid vs* will recreate the variable structure *vs* using only names that are not found in the *avoid* list of variables, and also renaming to avoid duplicate variable names in the structure. Variant names are found using *string_variant* (q.v.). If there are duplicates to be renamed, then the original name will be the rightmost in the variable structure.

Errors
```
3007  ?0 is not a term variable
4016  ?0 is not an allowed variable structure
```

Message 3007 applies to the avoid list, 27060 to the variable structure.

SML
```
val v_∃_intro : TERM -> THM -> THM ;
```

Description Introduce an existential quantified variable structure into a theorem.

Rule

$$\frac{\Gamma \vdash t[x,y,...]}{\Gamma \vdash \exists\ vs[x,y,...] \bullet\ t[x,y,...]} \qquad \begin{array}{l} v\_\exists\_intro \\ \ulcorner vs[x,y,...] \urcorner \end{array}$$

where ⌜vs[x,y,...]⌝ is a varstruct built from variables $\ulcorner x \urcorner$, $\ulcorner y \urcorner$, etc, which may contain duplicates.

Uses If the functionality is sufficient, this is superior in efficiency to both *∃_intro* and *simple_∃_intro* (q.v.).

Errors
```
4016  ?0 is not an allowed variable structure
```

SML
```
val ⇔_elim : THM -> (THM * THM);
```

Description Split a bi-implicative theorem into two implicative theorems.

Rule

$$\frac{\Gamma \vdash t1 \Leftrightarrow t2}{\Gamma \vdash t1 \Rightarrow t2;\ \Gamma \vdash t2 \Rightarrow t1} \qquad \Leftrightarrow\_elim$$

Errors
```
7062  ?0 is not of the form: ‘Γ ⊢ t1 ⇔ t2‘
```

SML
```
val ⇔_intro : THM -> THM -> THM;
```

Description Join two implicative theorems into an bi-implicative theorem.

Rule

$$\frac{\Gamma 1 \vdash t1 \Rightarrow t2;\ \Gamma 2 \vdash t1' \Rightarrow t2'}{\Gamma 1 \cup \Gamma 2 \vdash t1 \Leftrightarrow t2} \qquad \Leftrightarrow\_intro$$

where $t1$ and $t1'$ are α-convertible, as are $t2$ and $t2'$.

Errors
```
7040  ?0 is not of the form: ‘Γ ⊢ t1 ⇒ t2‘
7064  ?0 and ?1 are not of the form: ‘Γ ⊢ t1 ⇒ t2; Γ ⊢ t2a ⇒ t1a‘
      where ⌜t1⌝ and ⌜t1a⌝, ⌜t2⌝ and ⌜t2a⌝, are α−convertible
```

SML

```
val ⇔_match_mp_rule : THM -> THM -> THM;
```

Description A matching Modus Ponens for ⇔.

Rule

$$\frac{\Gamma 1 \vdash \forall\ x1\ ...\bullet\ t1\ \Leftrightarrow\ t2;\ \Gamma 2 \vdash t1'}{\Gamma 1' \cup \Gamma 2 \vdash t2'} \quad \Leftrightarrow\_match\_mp\_rule$$

where we type instantiate, generalise and specialise both conclusion and assumptions to get the first theorem's LHS to match the conclusion of the second theorem. Universal quantification, or the lack of it, in the first theorem makes no difference to the matching.

This may be partially evaluated with only first argument.

See Also *⇒_elim* (Modus Ponens on ⇒), *simple_⇔_match_mp_rule* *⇔_mp_rule* *⇔_match-_mp_rule1*

Errors

7044	*Cannot match ?0 and ?1*

SML

```
val ⇔_match_mp_rule1 : THM -> THM -> THM;
```

Description A matching Modus Ponens for ⇔ that doesn't affect assumption lists.

Rule

$$\frac{\Gamma 1 \vdash \forall\ x1\ ...\bullet\ t1\ \Leftrightarrow\ t2;\ \Gamma 2 \vdash t1'}{\Gamma 1 \cup \Gamma 2 \vdash t2'} \quad \Leftrightarrow\_match\_mp\_rule1$$

where $t1'$ is an instance of $t1$ under type instantiation and substitution for the x_i and the free variables of the first theorem, and where $t2'$ is the corresponding instance of $t2$. No type instantiation or substitution will occur in the assumptions of either theorem.

This may be partially evaluated with only first argument.

See Also *⇒_elim* (Modus Ponens on ⇒), *simple_⇔_match_mp_rule1*

Errors

7044	*Cannot match ?0 and ?1*
7046	*?0 is not of the form ‘Γ ⊢ ∀ x1 ... xn • u ⇔ v‘*

SML

```
val ⇔_mp_rule : THM -> THM -> THM;
```

Description This is reminiscent of Modus Ponens, but upon bi-implicative theorems.

Rule

$$\frac{\Gamma 1 \vdash t1\ \Leftrightarrow\ t2;\ \Gamma 2 \vdash t1'}{\Gamma 1 \cup \Gamma 2 \vdash t2} \quad \Leftrightarrow\_mp\_rule$$

where $t1$ and $t1'$ must be α-convertible.

A built-in inference rule.

See Also *⇒_elim* (true Modus Ponens, on ⇒), *⇔_match_mp_rule* (a "matching" version of *⇔_mp_rule*)

Errors

6024	*?0 and ?1 are not of the form: ‘Γ1 ⊢ t1 ⇔ t2‘ and ‘Γ2 ⊢ t1′‘ where ⌜t1⌝ and ⌜t1′⌝ are α−convertible*
6030	*?0 is not of the form: ‘Γ ⊢ t1 ⇔ t2‘*

SML

val **⇔_t_elim** : $THM \rightarrow THM$;

Description We can always eliminate $\ldots \Leftrightarrow T$.

Rule

$$\frac{\Gamma \vdash t \Leftrightarrow T}{\Gamma \vdash t} \quad \Leftrightarrow\_t\_elim$$

Errors

7106 ?0 not of the form '$\Gamma \vdash t \Leftrightarrow T$'

SML

val **⇔_t_intro** : $THM \rightarrow THM$;

Description The conclusion of a theorem is equal to T.

Rule

$$\frac{\Gamma \vdash t}{\Gamma \vdash t \Leftrightarrow T} \quad \Leftrightarrow\_t\_intro$$

SML

val **∧_intro** : $THM \rightarrow THM \rightarrow THM$;

Description Conjoin two theorems.

Rule

$$\frac{\Gamma 1 \vdash t1;\ \Gamma 2 \vdash t2}{\Gamma 1 \cup \Gamma 2 \vdash t1 \land t2} \quad \land\_intro$$

SML

val **∧_left_elim** : $THM \rightarrow THM$;

Description Give the left conjunct of a conjunction.

Rule

$$\frac{\Gamma \vdash t1 \land t2}{\Gamma \vdash t1} \quad \land\_left\_elim$$

Errors

7007 ?0 is not of the form: '$\Gamma \vdash t1 \land t2$'

SML

```
val ∧_rewrite_canon : THM -> THM list
val simple_¬_rewrite_canon : THM -> THM list
val ⇔_t_rewrite_canon : THM -> THM list
val f_rewrite_canon : THM -> THM list
val simple_∀_rewrite_canon : THM -> THM list
```

Description These are some of the standard canonicalisation functions used for breaking theorems up into lists of equations for use in rewriting. They perform the following transformations:

```
∧_rewrite_canon              (Γ ⊢ t1 ∧ t2)          = Γ ⊢ t1 ; Γ ⊢ t2
simple_¬_rewrite_canon       (Γ ⊢ ¬(t1 ∨ t2))       = (Γ ⊢ ¬t1 ∧ ¬t2)
simple_¬_rewrite_canon       (Γ ⊢ ¬∃x•t)            = (Γ ⊢ ∀x•¬t)
simple_¬_rewrite_canon       (Γ ⊢ ¬ ¬t)             = (Γ ⊢ t)
simple_¬_rewrite_canon       (Γ ⊢ ¬t)               = (Γ ⊢ t ⇔ F)
⇔_t_rewrite_canon        (Γ ⊢ t1 = t2)       = < failure >
⇔_t_rewrite_canon        (Γ ⊢ t)                    = (Γ ⊢ t ⇔ T)
f_rewrite_canon              (Γ ⊢ F)                       = (Γ ⊢ ∀ x • x)
simple_∀_rewrite_canon       (Γ ⊢ ∀x•t)             = Γ ⊢ t
```

Note that the functions whose names begin with *simple* do not handle paired quantifiers. Versions which do handle these quantifiers are also available.

See Also *¬_rewrite_canon*, *∀_rewrite_canon*.

Errors

```
26203 the conclusion of the theorem is already an equation
```

SML

```
val ∧_right_elim : THM -> THM;
```

Description Give the right conjunct of a conjunction.

Rule

$$\frac{\Gamma \vdash t1 \land t2}{\Gamma \vdash t2} \quad \land\_right\_elim$$

Errors

```
7007 ?0 is not of the form: ‘Γ ⊢ t1 ∧ t2‘
```

SML

```
val ∧_thm : THM;
```

Description Expanded form of definition of ∧

Theorem

$$\frac{}{\begin{array}{c}\forall\ t1\ t2 \bullet (t1 \land t2) \Leftrightarrow \\ (\forall\ b \bullet (t1 \Rightarrow t2 \Rightarrow b) \Rightarrow b)\end{array}} \quad \land\_thm$$

SML

```
val ∧_⇒_rule : THM -> THM;
```

Description A theorem whose conclusion is an implication from a conjunction is an equivalent to one whose conclusion is an implication of an implication.

Rule

$$\frac{\Gamma \vdash (a \land b) \Rightarrow c}{\Gamma \vdash a \Rightarrow b \Rightarrow c} \quad \land\_\Rightarrow\_rule$$

Errors

```
7009 ?0 is not of the form: ‘Γ ⊢ (a ∧ b) ⇒ c‘
```

SML

val ∨_**cancel_rule** : *THM −> THM −> THM*;

Description If we know a disjunction is true, and one of its disjuncts is false, then the other must be true. If the second theorem is the negation of both disjuncts, then the second disjunct will be eliminated. (modus tollendo ponens)

Rule

$$\frac{\Gamma 1 \vdash t1 \lor t2;\ \Gamma 2 \vdash \neg t1'}{\Gamma 1 \cup \Gamma 2 \vdash t2} \qquad \lor\_cancel\_rule$$

And:

Rule

$$\frac{\Gamma 1 \vdash t1 \lor t2;\ \Gamma 2 \vdash \neg t2'}{\Gamma 1 \cup \Gamma 2 \vdash t1} \qquad \lor\_cancel\_rule$$

where $t1'$ and $t1$ are α-convertible, as are $t2$ and $t2'$.

Errors

7010	*?0 is not of the form: ‘Γ ⊢ t1 ∨ t2‘*
7050	*?0 and ?1 are not of the form: ‘Γ1 ⊢ t1 ∨ t2‘ and ‘Γ2 ⊢ ¬t3‘ where ⌜t3⌝ is α−convertible to ⌜t1⌝ or ⌜t2⌝*

SML

val ∨_**elim** : *THM −> THM −> THM −> THM*;

Description Given a disjunctive theorem, and two further theorems, each containing one of the disjuncts in their assumptions, but with the same conclusion, we may eliminate the disjunct assumption from the second of the theorems.

Rule

$$\frac{\begin{array}{c}\Gamma 1 \vdash t1 \lor t2\\ \Gamma 2,\ t1' \vdash t\\ \Gamma 3,\ t2' \vdash t'\end{array}}{\Gamma 1 \cup \Gamma 2 \cup \Gamma 3 \vdash t} \qquad \lor\_elim$$

where $t1$ and $t1'$ are α-convertible, as are $t2$ and $t2'$, and t and t'. Actually, $t1'$ and $t2'$ do not have to be present in the assumption lists for this function to work.

Errors

7010	*?0 is not of the form: ‘Γ ⊢ t1 ∨ t2‘*
7083	*?0, ?1 and ?2 are not of the form: ‘Γ1 ⊢ t1 ∨ t2‘, ‘Γ2, t1a ⊢ t3‘ and ‘Γ3, t2a ⊢ t3a‘, where ⌜t1⌝ and ⌜t1a⌝, ⌜t2⌝ and ⌜t2a⌝, ⌜t3⌝ and ⌜t3a⌝ are each α−convertible*

SML

val ∨_**left_intro** : *TERM −> THM −> THM*;

Description Introduce a disjunct to the left of a theorem's conclusion.

Rule

$$\frac{\Gamma \vdash b}{\Gamma \vdash a \lor b} \qquad \begin{array}{l}\lor\_left\_intro\\ \ulcorner a \urcorner\end{array}$$

Errors

3031	*?0 is not of type ⌜:BOOL⌝*

SML

$val\ \lor\_$**right_intro** : $TERM\ {-}{>}\ THM\ {-}{>}\ THM;$

Description Introduce a disjunct to the right of a theorem's conclusion.

Rule

$$\frac{\Gamma \vdash b}{\Gamma \vdash b \lor a} \qquad \begin{array}{l}\lor\_right\_intro\\ \ulcorner a \urcorner\end{array}$$

Errors

3031	*?0 is not of type* $\ulcorner{:}BOOL\urcorner$

SML

$val\ \lor\_$**thm** : $THM;$

Description Expanded form of definition of ∨

Theorem

$$\frac{}{\begin{array}{c}\forall\ t1\ t2 \bullet (t1 \lor t2) \Leftrightarrow\\ (\forall\ b \bullet (t1 \Rightarrow b) \Rightarrow (t2 \Rightarrow b) \Rightarrow b)\end{array}} \qquad \lor\_thm$$

SML

$val\ \neg\_$**elim** : $TERM\ {-}{>}\ THM\ {-}{>}\ THM\ {-}{>}\ THM;$

Description Given two contradictory theorems with the same assumptions, conclude any other fact from the assumptions: input theorems may be in either order.

Rule

$$\frac{\Gamma 1 \vdash a\ ;\ \Gamma 2 \vdash \neg\ a}{\Gamma 1 \cup \Gamma 2 \vdash b} \qquad \begin{array}{l}\neg\_elim\\ \ulcorner b \urcorner\end{array}$$

Errors

3031	*?0 is not of type* $\ulcorner{:}BOOL\urcorner$
7004	*?0 and ?1 are not of the form*: '$\Gamma 1 \vdash a$' *and* '$\Gamma 2 \vdash \neg\ a$'

SML

$val\ \neg\_$**eq_sym_rule** : $THM\ {-}{>}\ THM\ ;$

Description If a is not equal to b then b is not equal to a.

Rule

$$\frac{\Gamma \vdash \neg(a = b)}{\Gamma \vdash \neg(b = a)} \qquad \neg\_eq\_sym\_rule$$

Errors

7091	*?0 is not of form*: '$\Gamma \vdash \neg(a = b)$'

SML

$val\ \neg\_$**intro** : $TERM\ {-}{>}\ THM\ {-}{>}\ THM\ {-}{>}\ THM;$

Description Given two theorems with contradictory conclusions (up to α-convertibility), their assumptions must be inconsistent, and thus any member of the lists (or indeed, anything else) may be proven false on the assumption of the remainder (reductio ad absurdum).

Rule

$$\frac{\Gamma 1 \vdash b\ ;\ \Gamma 2 \vdash \neg\ b}{(\Gamma 1 \cup \Gamma 2) \setminus \{a\} \vdash \neg\ a} \qquad \begin{array}{l}\neg\_intro\\ \ulcorner a \urcorner\end{array}$$

Works up to α-conversion, and input theorems may be in either order.

Errors

3031	*?0 is not of type* $\ulcorner{:}BOOL\urcorner$
7004	*?0 and ?1 are not of the form*: '$\Gamma 1 \vdash a$' *and* '$\Gamma 2 \vdash \neg\ a$'

SML

val **¬_simple_∀_conv** : *CONV*;

Description Move ¬ into a ∀ construct.

Rule

$$\frac{}{\vdash (\neg\ (\forall\ x \bullet t[x])) \Leftrightarrow \exists\ x \bullet \neg\ t[x]} \quad \begin{array}{l}\neg\_simple\_\forall\_conv \\ \ulcorner\neg\ (\forall\ x \bullet t[x])\urcorner\end{array}$$

This will work with any simple universal quantifier.

Errors

7036 ?0 not of the form: $\ulcorner\neg\ (\forall\ x \bullet t[x])\urcorner$

SML

val **¬_simple_∃_conv** : *CONV*;

Description Move ¬ into an ∃ construct.

Rule

$$\frac{}{\vdash (\neg\ (\exists\ x \bullet t[x])) \Leftrightarrow \forall\ x \bullet \neg\ t[x]} \quad \begin{array}{l}\neg\_simple\_\exists\_conv \\ \ulcorner\neg\ (\exists\ x \bullet t[x])\urcorner\end{array}$$

This will work with any simple existential quantifier.

Errors

7058 ?0 is not of the form: $\ulcorner\neg\ (\exists\ x \bullet t[x])\urcorner$
where $\ulcorner x\urcorner$ *is a variable*

SML

val **¬_thm1** : *THM*;

Description "Not t if and only if t is false."

Theorem

$$\frac{}{\vdash \forall\ t \bullet (\neg\ t) \Leftrightarrow (t \Leftrightarrow F)} \quad \neg\_thm1$$

SML

val **¬_thm** : *THM*;

Description Expanded form of definition of ¬:

Theorem

$$\frac{}{\forall\ t \bullet (\neg\ t) \Leftrightarrow (t \Rightarrow F)} \quad \neg\_thm$$

SML

val **¬_t_thm** : *THM*;

Description "Not true is false".

Theorem

$$\frac{}{\neg\ T \Leftrightarrow F} \quad \neg\_t\_thm$$

SML

val **¬_¬_conv** : *CONV*;

Description A double negation is redundant.

Conversion

$$\frac{}{\Gamma \vdash \neg\ (\neg\ t) \Leftrightarrow t} \quad \begin{array}{l}\neg\_\neg\_conv \\ \ulcorner\neg\ (\neg\ t)\urcorner\end{array}$$

Errors

7022 ?0 is not of the form: $\ulcorner\neg\ (\neg\ t)\urcorner$

SML

$val\ \neg\_\neg\_$**elim** : $THM\ {-}{>}\ THM$;

Description A double negation is redundant.

Rule

$$\frac{\Gamma \vdash \neg\ (\neg\ t)}{\Gamma \vdash t} \quad \neg\_\neg\_elim$$

Errors

7006 ?0 is not of the form: '$\Gamma \vdash \neg\ (\neg\ t)$'

SML

$val\ \neg\_\neg\_$**intro** : $THM\ {-}{>}\ THM$;

Description We may always introduce a double negation.

Rule

$$\frac{\Gamma \vdash t}{\Gamma \vdash \neg\ (\neg\ t)} \quad \neg\_\neg\_intro$$

SML

$val\ \neg\_\forall\_$**conv** : $CONV$;

Description Move $\neg$ into a $\forall$ construct.

Rule

$$\frac{}{\vdash (\neg\ (\forall\ x\bullet\ t[x])) \Leftrightarrow \exists\ x\bullet\ \neg\ t[x]} \quad \begin{array}{l} \neg\_\forall\_conv \\ \ulcorner\neg\ (\forall\ x\bullet\ t[x])\urcorner \end{array}$$

See Also $\neg\_simple\_\forall\_conv$ which only works with simple $\forall$-abstractions, $\neg\_\exists\_conv$

Errors

27019 ?0 not of the form: $\ulcorner\neg\ (\forall\ x\bullet\ t[x])\urcorner$
where $\ulcorner x\urcorner$ *is a varstruct*

SML

$val\ \neg\_\forall\_$**thm** : THM;

Description Used in pushing negations through simple universal quantifications.

Theorem

$$\frac{}{\vdash \forall\ p\bullet\ \neg\ \$\forall\ p \Leftrightarrow (\exists\ x\bullet\ \neg\ p\ x)} \quad \neg\_\forall\_thm$$

SML

$val\ \neg\_\exists\_$**conv** : $CONV$;

Description Move $\neg$ into an $\exists$ construct.

Rule

$$\frac{}{\vdash (\neg\ (\exists\ x\bullet\ t[x])) \Leftrightarrow \forall\ x\bullet\ \neg\ t[x]} \quad \begin{array}{l} \neg\_\exists\_conv \\ \ulcorner\neg\ (\exists\ x\bullet\ t[x])\urcorner \end{array}$$

See Also $\neg\_simple\_\exists\_conv$ which only works with simple $\exists$-abstractions, $\neg\_\forall\_conv$

Errors

27020 ?0 is not of the form: $\ulcorner\neg\ (\exists\ x\bullet\ t[x])\urcorner$
where $\ulcorner x\urcorner$ *is a varstruct*

SML

$val\ \neg\_\exists\_\mathbf{thm} : THM;$

Description Used in pushing negations through simple existential quantifications.

Theorem

$$\frac{}{\vdash \forall\ p\bullet\ \neg\ \$\exists\ p \Leftrightarrow (\forall\ x\bullet\ \neg\ p\ x)} \quad \neg\_\exists\_thm$$

SML

$val \Rightarrow\_\mathbf{elim} : THM \ {-}{>}\ THM \ {-}{>}\ THM;$
$val \Rightarrow\_\mathbf{mp\_rule} : THM \ {-}{>}\ THM \ {-}{>}\ THM;$

Description Modus Ponens (which is why we introduce the alias $\Rightarrow\_mp\_rule$, though $\Rightarrow\_elim$ is shorter, conventional, and the preferred name).

Rule

$$\frac{\Gamma 1 \vdash t1 \Rightarrow t2;\ \Gamma 2 \vdash t1'}{\Gamma 1 \cup \Gamma 2 \vdash t2} \quad \Rightarrow\_elim$$

where $t1$ and $t1'$ must be α-convertible. A primitive inference rule.

See Also $\Leftrightarrow\_mp\_rule$(Modus Ponens on $\Leftrightarrow$), $\Rightarrow\_match\_mp\_rule$ (a "matching" version of this function).

Errors

6010 *?0 is not of the form: '$\Gamma \vdash t1 \Rightarrow t2$'*
6011 *?0 and ?1 are not of the forms: '$\Gamma 1 \vdash t1 \Rightarrow t2$' and '$\Gamma 2 \vdash t1'$' where ⌜t1⌝ and ⌜t1'⌝ are α−convertible*

SML

$val \Rightarrow\_\mathbf{intro} : TERM \ {-}{>}\ THM \ {-}{>}\ THM;$

Description Prove an implicative theorem, removing, if α-convertibly present, the antecedent of the implication from the assumption list.

Rule

$$\frac{\Gamma \vdash t2}{\Gamma - \{t1\} \vdash t1 \Rightarrow t2} \quad \begin{array}{l}\Rightarrow\_intro \\ \ulcorner t1 \urcorner\end{array}$$

A primitive inference rule.

See Also *disch_rule* (which fails if term not in assumption list)

Errors

3031 *?0 is not of type ⌜:BOOL⌝*

SML
```
val ⇒_match_mp_rule : THM -> THM -> THM ;
```

Description A matching Modus Ponens rule for an implicative theorem.

Rule

$$\frac{\Gamma 1 \vdash \forall\ x1\ ...\bullet\ t1 \Rightarrow t2;\ \Gamma 2 \vdash t1'}{\Gamma 1' \cup \Gamma 2 \vdash t2'} \qquad \Rightarrow\_match\_mp\_rule$$

where we type instantiate, generalise and specialise to get the first theorem's antecedent to match the conclusion of the second theorem. Universal quantification, or the lack of it, in the first theorem makes no difference to the matching.

This may be partially evaluated with only the first argument.

See Also *⇒_match_mp_rule1*, *⇒_elim*

Errors

7044	*Cannot match ?0 and ?1*

SML
```
val ⇒_match_mp_rule1 : THM -> THM -> THM ;
val ⇒_match_mp_rule2 : THM -> THM -> THM ;
```

Description Two variants of a matching Modus Ponens rule for an implicative theorem.

Rule

$$\frac{\Gamma 1 \vdash \forall\ x1\ ...\bullet\ t1 \Rightarrow t2;\ \Gamma 2 \vdash t1'}{\Gamma 1 \cup \Gamma 2 \vdash t2'} \qquad \Rightarrow\_match\_mp\_rule$$

where $t1'$ is an instance of $t1$ under type instantiation and substitution for the x_i and the free variables of the first theorem, and where $t2'$ is the corresponding instance of $t2$. The type instantiations and substitutions are obtained by matching $t1$ and $t1'$ using *term_match*.

⇒_match_mp_rule2 is just like *⇒_match_mp_rule1* except that the instantiations and substitutions returned by *term_match* are extended to replace type variables that do not occur in $t1$ or in $\Gamma 1$ and x_i that do not occur free in $t1$ by fresh variables to avoid clashes with each other and with the type variables and free variables of $\Gamma 1$ and $\Gamma 2$.

Types in the assumptions of the theorems will not be instantiated.

Both rules may be partially evaluated with only the first argument.

Errors

7044	*Cannot match ?0 and ?1*
7045	*?0 is not of the form '$\Gamma \vdash \forall\ x1\ ...\ xn \bullet u \Rightarrow v$'*

SML
```
val ⇒_trans_rule : THM -> THM -> THM;
```

Description Transitivity of ⇒.

Rule

$$\frac{\Gamma 1 \vdash t1 \Rightarrow t2;\ \Gamma 2 \vdash t2' \Rightarrow t3}{\Gamma 1 \cup \Gamma 2 \vdash t1 \Rightarrow t3} \qquad \Rightarrow\_trans\_rule$$

where $t2$ and $t2'$ are α-convertible.

Errors

7040	*?0 is not of the form: '$\Gamma \vdash t1 \Rightarrow t2$'*
7042	*?0 and ?1 are not of the form: '$\Gamma 1 \vdash t1 \Rightarrow t2$' and '$\Gamma 2 \vdash t2a \Rightarrow t3$' where ⌜t2⌝ and ⌜t2a⌝ are α−convertible*

SML

$val \Rightarrow\_\wedge\_$**rule** : THM −> THM;

Description A theorem whose conclusion is an implication of an implication is equivalent to one whose conclusion is a conjunction and an implication.

Rule

$$\frac{\Gamma \vdash a \Rightarrow b \Rightarrow c}{\Gamma \vdash (a \wedge b) \Rightarrow c} \qquad \Rightarrow\_\wedge\_rule$$

Errors

7008 ?0 is not of the form: ‘$\Gamma \vdash a \Rightarrow b \Rightarrow c$‘

SML

$val\ \forall\_$**arb_elim** : THM −> THM;

Description Specialise a universally quantified theorem with a machine generated variable or variable structure.

Rule

$$\frac{\Gamma \vdash \forall\ vs[x,y,...]\bullet\ p[x,y,...]}{\Gamma \vdash p[x',y',...]} \qquad \forall\_arb\_elim$$

where x', y', etc, are not variables (free or bound) in p or Γ, created by gen_vars(q.v).

See Also $\forall\_elim$

Errors

27011 ?0 is not of the form: ‘$\Gamma \vdash \forall\ x\bullet\ t$‘ *where* $\ulcorner x \urcorner$ *is a varstruct*

SML

$val\ \forall\_$**asm_rule** : $TERM$ −> $TERM$ −> THM −> THM;

Description Generalise an assumption (Left ∀ introduction).

Rule

$$\frac{\Gamma,\ p'[x] \vdash q[x]}{\Gamma,\ \forall\ x\bullet\ p'[x] \vdash q[x]} \qquad \begin{array}{l}\forall\_asm\_rule\\ \ulcorner x \urcorner\\ \ulcorner p[x] \urcorner\end{array}$$

where p and p' are α-convertible. x **may** be free in Γ. The function will work even if $p'[x]$ is not present in the assumption list.

Errors

4016 ?0 is not an allowed variable structure

SML

val $\forall$_**elim** : $TERM$ −> THM −> THM;

Description Specialise a universally quantified theorem with a given value, instantiating the type of the theorem as necessary.

Rule

$$\frac{\Gamma \vdash \forall\ x\bullet\ t2[x]}{\Gamma \vdash\ t2'[t1]} \qquad \begin{array}{l}\forall\_elim\\ \ulcorner t1 \urcorner\end{array}$$

where $t2'$ is renamed from $t2$ to prevent bound variable capture and possibly type instantiated, and x is a varstruct, instantiable to the structure of $t1$. The value $t1$ will be expanded using Fst and Snd as necessary to match the structure of $\ulcorner x \urcorner$.

See Also *list_∀_elim*, *all_∀_elim*.

Errors

27011 ?0 is not of the form: ‘$\Gamma \vdash \forall\ x\bullet\ t$‘ where $\ulcorner x \urcorner$ is a varstruct
27012 ?0 is not of the form: ‘$\Gamma \vdash \forall\ x\bullet\ t$‘ where the type of $\ulcorner x \urcorner$
is instantiable to the type of ?1
27013 ?0 is not of the form: ‘$\Gamma \vdash \forall\ x\bullet\ t$‘ where the type of $\ulcorner x \urcorner$
is instantiable to the type of ?1 without instantiating
type variables in the assumptions

SML

val $\forall$_**intro** : $TERM$ −> THM −> THM;

Description Introduce a universally quantified theorem.

Rule

$$\frac{\Gamma \vdash\ t}{\Gamma \vdash \forall\ x'\bullet\ t} \qquad \begin{array}{l}\forall\_intro\\ \ulcorner x \urcorner\end{array}$$

Where $\ulcorner x' \urcorner$ is an allowed variable structure based on $\ulcorner x \urcorner$, but with duplicate variables renamed, the original name being rightmost in the resulting variable structure.

See Also *list_∀_intro*, *all_∀_intro*.

Errors

4016 ?0 is not an allowed variable structure
6005 ?0 occurs free in assumption list

SML

$val\ \forall\_\textbf{reorder\_conv} : TERM\ {-}{>}\ CONV;$

Description Reorder universal quantifications.

Rule

$$\frac{}{(\forall\ y1\ ..\ ym\bullet\ t2) \Leftrightarrow (\forall\ x1\ ..\ xn\bullet\ t1)} \quad \begin{array}{l} \forall\_reorder\_conv \\ \ulcorner\forall\ x1\ ..\ xn\bullet\ t1\urcorner \\ \ulcorner\forall\ y1\ ..\ ym\bullet\ t2\urcorner \end{array}$$

where the x_i and y_i are varstructs, and the reordering, restructuring (by pairing) and renaming requested is provable by this function. The presence of redundant quantifiers, including duplicates, is also handled.

Example

```
:> ∀_reorder_conv ⌜∀ (x,q) z• x ∧ z⌝ ⌜∀ (z,z,y) x• x ∧ z⌝;
val it = ⊢ (∀ (z, z, y) x• x ∧ z) ⇔ (∀ (x, q) z• x ∧ z) : THM
Note that before more sophisticated attempts, the conversion
will try $α\_conv$ on the two term arguments.
```

See Also $\exists\_reorder\_conv$

Errors

27050 *Cannot prove equality of ?0 and ?1*

SML

$val\ \forall\_\textbf{uncurry\_conv} : CONV;$

Description Convert a paired universally quantified term into simple universal quantifications of the same term.

Conversion

$$\frac{}{\begin{array}{c}\Gamma \vdash \forall\ vs[x,y,...]\bullet\ f[x,y,...] = \\ \forall\ x\ y\ ...\bullet\ f[x,y,...]\end{array}} \quad \begin{array}{l} \forall\_uncurry\_conv \\ \ulcorner\forall\ vs[x,y,...]\bullet\ f[x,y,...]\urcorner \end{array}$$

where $vs[x, y, ...]$ is an allowed variable structure with variables $x, y, ...$ It may not be a simple variable.

See Also $\lambda\_varstruct\_conv$, $all\_\forall\_uncurry\_conv$.

Errors

27038 *?0 is not of the form:* $\ulcorner\forall\ (x,y)\bullet\ f\urcorner$

SML

$val\ \forall\_\Leftrightarrow\_\textbf{rule} : TERM\ {-}{>}\ THM\ {-}{>}\ THM;$

Description Universally quantify a variable on both sides of an equivalence.

Rule

$$\frac{\Gamma \vdash p[x] \Leftrightarrow q[x]}{\Gamma \vdash (\forall\ x\bullet\ p[x]) \Leftrightarrow (\forall\ x\bullet\ q[x])} \quad \begin{array}{l} \forall\_\Leftrightarrow\_rule \\ \ulcorner x\urcorner \end{array}$$

where x is a varstruct.

Errors

6005 *?0 occurs free in assumption list*
6020 *?0 is not of the form:* ‘$\Gamma \vdash t1 = t2$‘
7062 *?0 is not of the form:* ‘$\Gamma \vdash t1 \Leftrightarrow t2$‘
4016 *?0 is not an allowed variable structure*

SML

```
val ∃_asm_rule : TERM -> TERM -> THM -> THM;
```

Description Existentially quantify an assumption (Left ∃ introduction).

Rule

$$\frac{\Gamma,\ p'[x] \vdash q}{\Gamma,\ \exists\ x \bullet\ p'[x] \vdash q} \qquad \begin{array}{l} \exists\_asm\_rule \\ \ulcorner x \urcorner \\ \ulcorner p[x] \urcorner \end{array}$$

where p and p' are α-convertible. where the variables of the varstruct x are not free in Γ or q. The assumption need not be present for the rule to apply.

Errors

3015 ?1 is not of type ⌜:BOOL⌝
4016 ?0 is not an allowed variable structure
6005 ?0 occurs free in assumption list
27052 ?0 has members appearing free in ?1 other than in assumption ?2

Message 3015 is just passed on from low level functions, which is why it has "?1" not "?0".

SML

```
val ∃_elim : TERM -> THM -> THM -> THM;
```

Description Eliminate an existential quantifier by reference to an arbitrary varstruct satisfying the predicate.

Rule

$$\frac{\begin{array}{c}\Gamma 1 \vdash \exists\ vs[x1,x2,...] \bullet\ t1[x1,x2,...]; \\ \Gamma 2,\ t1[y1,y2,...] \vdash t2\end{array}}{\Gamma 1 \cup \Gamma 2 \vdash t2} \qquad \begin{array}{l} \exists\_elim \\ \ulcorner vs[y1,y2,...] \urcorner \end{array}$$

$t1[y1, y2, ...]$ need not actually be present in the assumptions of the second theorem. The y_i must be free variables, none of whom are present elsewhere in the second theorem, or in the conclusion of the first. The y_i may contain duplicates as long as the end pattern matches the x_i in required duplicates. The term argument may be a less complex variable structure than the bound variable structure of the theorem, as *Fst* and *Snd* are used to make them match. For example, the following rule holds true:

Rule

$$\frac{\begin{array}{c}\Gamma 1 \vdash \exists\ (p,q) \bullet\ t1[p,q]; \\ \Gamma 2,\ t1[Fst\ x,\ Snd\ x] \vdash t2\end{array}}{\Gamma 1 \cup \Gamma 2 \vdash t2} \qquad \begin{array}{l} \exists\_elim \\ \ulcorner x \urcorner \end{array}$$

Errors

27042 ?0 does not match the bound varstruct of ?1
27046 ?0 is not of the form ‘Γ ⊢ ∃ vs• t‘
27051 ?0 has members appearing free in conclusion of ?1
27052 ?0 has members appearing free in ?1 other than in assumption ?2

SML

```
val ∃_intro_thm : THM;
```

Description Introduction of existential quantification.

Theorem

$$\frac{}{\vdash \forall\ P\ x \bullet\ P\ x \Rightarrow \$\exists\ P} \qquad \exists\_intro\_thm$$

SML
```
val ∃_intro : TERM -> THM -> THM ;
```

Description Introduce an existential quantifier by reference to a witness.

Rule

$$\frac{\Gamma \vdash t[t1,t2,...]}{\Gamma \vdash \exists\ vs[x',y',...]\bullet\ t[x,y,...]} \quad \begin{array}{l} \exists\_intro \\ \ulcorner\exists\ vs[x,y,...]\bullet\ t[x,y,...]\urcorner \end{array}$$

where ⌜vs[x,y,...]⌝ is varstruct built from variables $\ulcorner x\urcorner$, $\ulcorner y\urcorner$, etc, and the $\ulcorner x'\urcorner$ are renamed if duplicated inside the varstruct, all but the rightmost being so renamed.

Errors
```
4020  ?0 is not of form: ⌜∃ vs• t⌝
7047  ?0 cannot be matched to conclusion of theorem ?1
```

SML
```
val ∃_reorder_conv : TERM -> CONV;
```

Description Reorder existential quantifications.

Rule

$$\frac{}{(\exists\ y1\ ..\ ym\bullet\ t2) \Leftrightarrow (\exists\ x1\ ..\ xn\bullet\ t1)} \quad \begin{array}{l} \exists\_reorder\_conv \\ \ulcorner\exists\ x1\ ..\ xn\bullet\ t1\urcorner \\ \ulcorner\exists\ y1\ ..\ ym\bullet\ t2\urcorner \end{array}$$

where the x_i and y_i are varstructs, and the reordering, restructuring (by pairing) and renaming requested is provable by this function. The presence of redundant quantifiers, including duplicates, is also handled.

Example
```
:> ∃_reorder_conv ⌜∃ (x,q) z• x ∧ z⌝ ⌜∃ (z,z,y) x• x ∧ z⌝;
val it = ⊢ (∃ (z, z, y) x• x ∧ z) ⇔ (∃ (x, q) z• x ∧ z) : THM
Note that before more sophisticated attempts, the conversion
will try $α\_conv$ on the two term arguments.
```

See Also ∀_reorder_conv

Errors
```
27050  Cannot prove equality of ?0 and ?1
```

SML
```
val ∃_uncurry_conv : CONV;
```

Description Convert a paired existentially quantified term into simple universal quantifications of the same term.

Conversion

$$\frac{}{\begin{array}{l}\vdash \exists\ vs[x,y,...]\bullet\ f[x,y,...] = \\ \quad \exists\ x\ y\ ...\bullet\ f[x,y,...]\end{array}} \quad \begin{array}{l} \exists\_uncurry\_conv \\ \ulcorner\exists\ vs[x,y,...]\bullet\ f[x,\ y,...]\urcorner \end{array}$$

where $vs[x, y, ...]$ is an allowed variable structure with variables $x, y, ...$ It may not be a simple variable.

See Also λ_varstruct_conv, all_∃_uncurry_conv, ∀_uncurry_conv.

Errors
```
27047 ?0 is not of the form: ⌜∃ (x,y)• f⌝
```

SML

$val\ \exists\_\epsilon\_\mathbf{conv} : CONV;$

Description Give that ϵ of a predicate satisfies the predicate by reference to an $\exists$ construct. It can properly handle paired existence.

Rule

$$\frac{}{\Gamma \vdash (\exists\ x\bullet\ p[x]) = p(\epsilon\ x\bullet\ p\ x)} \qquad \begin{array}{l}\exists\_\epsilon\_conv\\ \ulcorner\exists\ x\bullet\ p[x]\urcorner\end{array}$$

If x is formed by paired then the Fst and Snd are used to extract the appropriate bits of the ϵ-term for distribution in $p[\epsilon\ x\bullet\ p\ x]$.

See Also $\exists\_\epsilon\_rule$

Errors

27024 ?0 is not of the form: '$\Gamma \vdash \exists\ x\bullet\ p[x]$'
where $\ulcorner x\urcorner$ *is a varstruct*

SML

$val\ \exists\_\epsilon\_\mathbf{rule} : THM\ {->}\ THM;$

Description Give that ϵ of a predicate satisfies the predicate by reference to an $\exists$ construct. It can properly handle paired existence.

Rule

$$\frac{\Gamma \vdash \exists\ x\bullet\ p[x]}{\Gamma \vdash p[\epsilon\ x\bullet\ p\ x]} \qquad \exists\_\epsilon\_rule$$

If x is formed by paired then the Fst and Snd are used to extract the appropriate bits of the ϵ-term for distribution in $p[\epsilon\ x\bullet\ p\ x]$.

See Also $\exists\_\epsilon\_conv$

Errors

27024 ?0 is not of the form: '$\Gamma \vdash \exists\ x\bullet\ p[x]$'
where $\ulcorner x\urcorner$ *is a varstruct*

SML

$val\ \exists_{\mathbf{1}}\_\mathbf{conv} : CONV;$

Description This is a conversion which turns a unique existential quantifier into an equivalent existential quantifier

Conversion

$$\frac{}{\begin{array}{c}\vdash (\exists_1 vs[x1,...]\bullet t[x1,...]) \Leftrightarrow\\ (\exists vs[x1,...]\bullet t[x1,...] \wedge\\ \forall vs[x1',...]\bullet t[x1',...] \Rightarrow\\ vs[x1',...] = vs[x1,...])\end{array}} \qquad \begin{array}{l}\exists_1\_conv\\ \ulcorner\exists_1 vs[x1,...]\bullet t[x1,...]\urcorner\end{array}$$

Uses Tactic and conversion programming.

See Also $strip\_tac$, $simple\_\exists\_1\_conv$

Errors

27053 ?0 is not of the form: $\ulcorner\exists_1 vs\bullet t\urcorner$

SML

```
val ∃₁_elim : THM -> THM;
```

Description Express a ∃_1 in terms of ∃ and a uniqueness property.

Rule

$$\frac{\Gamma \vdash \exists_1\ vs[a,b,...]\bullet\ P[a,b,...]}{\begin{array}{c}\Gamma \vdash \exists\ vs[a,b,...]\bullet\ P[a,b,...] \land \\ \forall\ vs[a',b',...]\bullet\ P[x1,x2,...] \\ \Rightarrow \\ vs[a',b',...] = vs[a,b,...]\end{array}} \quad \exists_1\_elim$$

where the a', etc, are variants of the a.

Errors

27022 ?0 is not of the form: '$\Gamma \vdash \exists_1\ x\bullet\ P[x]$'
where ⌜x⌝ is a varstruct

SML

```
val ∃₁_intro : THM -> THM -> THM;
```

Description Introduce ∃_1 by reference to a witness, and a uniqueness theorem.

Rule

$$\frac{\begin{array}{c}\Gamma 1 \vdash P'[t'] \\ \Gamma 2 \vdash \forall\ x\bullet\ P[x] \Rightarrow x = t\end{array}}{\Gamma 1 \cup \Gamma 2 \vdash \exists_1\ x\bullet\ P[x]} \quad \exists_1\_intro$$

Where P' is α-convertible to P, and t' is α-convertible to t. Notice that for the resulting theorem we take the varstruct, x, and the form of the predicate, P, from the second theorem.

Errors

27021 ?0 and ?1 are not of the form: '$\Gamma 1 \vdash Pa[ta]$' and
'$\Gamma 2 \vdash \forall\ vs[x,y,..]\bullet\ P[x,y...] \Rightarrow vs[x,y,..] = t$'
where ⌜Pa⌝ and ⌜P⌝, ⌜ta⌝ and ⌜t⌝ are $\alpha-$convertible
and ⌜x⌝ is a varstruct
27054 ?0 not of the form: '$\Gamma \vdash \forall\ vs[x,y,..]\bullet\ P[x,y...] \Rightarrow vs[x,y,..] = t$'

SML

```
val ∃₁_thm : THM;
```

Description Expanded form of definition of $\exists_1$

Theorem

$$\frac{}{\begin{array}{c}\vdash \forall\ P \bullet (\$\exists_1\ P) \Leftrightarrow \\ (\exists\ t \bullet (P\ t) \land \\ (\forall\ x \bullet (P\ x) \Rightarrow x = t))\end{array}} \quad \exists_1\_thm$$

SML

```
val α_conv : TERM -> CONV;
```

Description Returns a theorem that two terms are equal, should they be α-convertible.

Rule

$$\frac{}{\vdash t1 = t2} \quad \begin{array}{l}\alpha\_conv \\ ⌜t2⌝ \\ ⌜t1⌝\end{array}$$

Errors

3012 ?0 and ?1 do not have the same types
7034 ?0 and ?1 are not $\alpha-$convertible

SML
```
val β_conv : CONV;
```

Description Apply a β-reduction to an abstraction.

Rule

$$\frac{}{\vdash ((\lambda\ x\bullet\ t[x])y)\ =\ t'[y]} \qquad \begin{array}{l}\beta\_conv \\ \ulcorner(\lambda\ x\bullet\ t1[x])t2\urcorner\end{array}$$

where x may be any varstruct allowed by the ICL HOL syntax, y is an instance of this structure, and t' is α-convertible to t, changed to avoid variable capture.

When the bound variable structure has a pair, where the value applied to does not, then *Fst* and *Snd* are introduced as necessary, e.g.:

Example
```
β_conv ⌜(λ (x,y)• f x y) p⌝ =
        ⊢ (λ (x,y)• f x y) p = f (Fst p) (Snd p)
```

See Also *simple_β_conv*, *β_rule*

Errors
```
27008 ?0 is not of the form: ⌜(λ x• t1[x])t2⌝
      where ⌜x⌝ is a varstruct
```

SML
```
val β_rule : THM -> THM;
```

Description An elimination rule for λ, which can handle paired abstractions.

Rule

$$\frac{\Gamma \vdash (\lambda\ x\bullet\ t[x])\ y}{\Gamma \vdash t[y]} \qquad \beta\_rule$$

See Also *β_conv*

Errors
```
27007 ?0 is not of the form: ‘Γ ⊢ (λ x• t[x]) y‘
      where ⌜x⌝ is a varstruct
```

SML

$val\ \epsilon\_\mathbf{elim\_rule} : TERM\ {-}{>}\ THM\ {-}{>}\ THM\ {-}{>}\ THM;$

Description Given that ϵ of a predicate satisfies that predicate, then in a different theorem we may eliminate an assumption that claims an otherwise unused variable structure satisfies the predicate.

Rule

$$\frac{\Gamma 1 \vdash t'\ (\$\epsilon\ t'');\quad \Gamma 2,\ t\ vs \vdash s}{\Gamma 1 \cup \Gamma 2 \vdash s} \quad \begin{array}{l}\epsilon\_elim\_rule\\ \ulcorner vs \urcorner\end{array}$$

where t, t' and t'' are α-convertible, and vs is a varstruct, with no duplicates, and with its free variables occurring nowhere else in the second theorem, or in the conclusion of the first. In fact, $(\$\epsilon\ t'')$ here can be any term, it is not constrained to be an application of the choice function.

Errors

4016	*?0 is not an allowed variable structure*
7019	*?0 is not of the form:* $`\Gamma \vdash t1(\epsilon\ t1)`$
7054	*?0 is not of same type as choice sub−term of first theorem*
27043	*?0 is repeated in the varstruct ?1*
27045	*Arguments ?0; ?1 and ?2 not of the form* $\ulcorner vs \urcorner$; $`\Gamma 1 \vdash t\ (\epsilon\ t)`$ *and* $`\Gamma 2,\ (t\ vs) \vdash s`$
27051	*?0 has members appearing free in conclusion of ?1*
27052	*?0 has members appearing free in ?1 other than in assumption ?2*

SML

$val\ \epsilon\_\mathbf{intro\_rule} : THM\ {-}{>}\ THM;$

Description Given a theorem whose conclusion is a function application, we know that the "function" is a predicate, and the rule states that ϵ of this predicate will satisfy the predicate.

Rule

$$\frac{\Gamma \vdash t1\ \ t2}{\Gamma \vdash t1\ (\$\epsilon\ t1)} \quad \epsilon\_intro\_rule$$

Errors

7016	*?0 is not of the form:* $`\Gamma \vdash t1\ \ t2`$

SML

$val\ \eta\_\mathbf{conv} : CONV;$

Description The rule for η conversion.

Conversion

$$\frac{}{\vdash (\lambda\ vs\bullet\ t\ vs) = t} \quad \begin{array}{l}\eta\_conv\\ \ulcorner \lambda\ vs\bullet\ t\ vs \urcorner\end{array}$$

where t contains no free instances of the variables of varstruct vs.

Errors

27018	*?0 is not of the form:* $\ulcorner \lambda\ vs\bullet\ t\ vs' \urcorner$ *where* $\ulcorner vs \urcorner$ *is a varstruct*
27023	*?0 is not of the form:* $\ulcorner \lambda\ vs\bullet\ t\ vs \urcorner$ *where* $\ulcorner t \urcorner$ *should not contain* $\ulcorner vs \urcorner$

SML

$val\ \lambda\_\mathbf{C} : CONV \ {->}\ CONV;$

Description Apply a conversion to the body of an abstraction:

Rule

$$\frac{}{\vdash (\lambda\ x\bullet\ p[x]) = (\lambda\ x\bullet\ pa[x])} \qquad \begin{array}{l}\lambda\_C\\ (c : CONV)\\ \ulcorner\lambda\ x\bullet\ p\urcorner\end{array}$$

where $c\ p[x]$ gives '$\vdash p[x] = pa[x]$'.

Errors

4002	*?0 is not of form:* $\ulcorner\lambda\ vs\bullet\ t\urcorner$
7104	*Result of conversion, ?0, ill−formed*

Also as the failure of the conversion.

SML

$val\ \lambda\_\mathbf{eq\_rule} : TERM \ {->}\ THM \ {->}\ THM;$

Description Given an equational theorem, return the equation formed by abstracting the term argument (which must be an allowed variable structure) from both sides.

Rule

$$\frac{\Gamma \vdash t1[x] = t2[x]}{\Gamma \vdash (\lambda\ x\bullet\ t1[x]) = (\lambda\ x\bullet\ t2[x])} \qquad \begin{array}{l}\lambda\_eq\_rule\\ \ulcorner x\urcorner\end{array}$$

Errors

4016	*?0 is not an allowed variable structure*
6005	*?0 occurs free in assumption list*
6020	*?0 is not of the form:* '$\Gamma \vdash t1 = t2$'

SML

$val\ \lambda\_\mathbf{pair\_conv} : CONV;$

Description This conversion eliminates abstraction over pairs in favour of abstraction over elements of pairs. The bound variables of the resulting λ-abstraction do not have pair types.

Rule

$$\frac{}{\begin{array}{c}\vdash (\lambda\ v\bullet\ t) =\\ (\lambda\ (v1,\ v2)\bullet\ t'[(v1,\ v2)/v])\end{array}} \qquad \begin{array}{l}\lambda\_pair\_conv\\ \ulcorner\lambda\ v{:}'a \times 'b\bullet\ t\urcorner\end{array}$$

Rule

$$\frac{}{\begin{array}{l}\vdash (\lambda\ (v,\ w)\bullet\ t) =\\ (\lambda\ ((v1,\ v2),\ (w1,\ w2))\bullet\\ \quad t[(v1,\ v2)/v,\ (w1,\ w2)/v])\end{array}} \qquad \begin{array}{l}\lambda\_pair\_conv\\ \ulcorner\lambda\ (v,\ w){:}('a \times 'b) \times ('c \times 'd)\bullet\ t\urcorner\end{array}$$

and so on.

Errors

27055	*The type of ?0 is not of the form* $\sigma \times \tau$

SML
```
val λ_rule : TERM -> THM -> THM;
```

Description An introduction rule for λ:

Rule

$$\frac{\Gamma \vdash s[t]}{\Gamma \vdash (\lambda\ x \bullet s[x])\ t} \qquad \begin{array}{l} \lambda\_rule \\ \ulcorner t \urcorner \end{array}$$

where x is a machine generated variable.

SML
```
val λ_varstruct_conv : TERM -> CONV;
```

Description This conversion is a generalisation of *α_conv* allowing one to convert a λ-abstraction into an equivalent λ-abstraction that differs only in the form of the varstruct and the corresponding use of *Fst* in the *Snd* in the body of the abstraction.

Rule

$$\frac{}{\begin{array}{c}\vdash (\lambda\ vs2[x2,y2,...]\bullet\ t[x2,y2,...]) = \\ (\lambda\ vs1[x1,y1,...]\bullet\ t'[x1,y1,...])\end{array}} \qquad \begin{array}{l} \lambda\_varstruct\_conv \\ \ulcorner \lambda\ vs1[x1,y1,...]\bullet\ t[x1,y1,...] \urcorner \\ \ulcorner \lambda\ vs2[x2,y2,...]\bullet\ t'[x2,y2,...] \urcorner \end{array}$$

Where the types of $vs1[x1, y1, ...]$ and $vs2[x2, y2, ...]$ are the same, and t' and t differ only in applications of *Fst* and *Snd* to the bound variables.

For example,

Rule

$$\frac{}{\begin{array}{c}\vdash ((\lambda\ x\bullet\ Fst\ x + Snd\ x = 1) = \\ (\lambda\ (a,\ b)\bullet\ a + b = 1)\end{array}} \qquad \begin{array}{l} \lambda\_varstruct\_conv \\ \ulcorner \lambda\ (a,\ b)\bullet\ a + b = 1 \urcorner \\ \ulcorner \lambda\ x\bullet\ Fst\ x + Snd\ x = 1 \urcorner \end{array}$$

See Also *α_conv* for a more limited form of renaming.

Errors
```
27050 Cannot prove equality of ?0 and ?1
```

8.2 Subgoal Package

SML
| *signature* **SubgoalPackage** = *sig*

Description This provides the subgoal package, which provides an interactive backward proof mechanism, based on the application of tactics.

Errors

30009 There are no goals to prove
30017 Label ?0 has no corresponding goal
30023 ?0 cannot be interpreted as a goal
30028 Label may not contain ?0, as less than 1
30041 Label ?0 has been superseded
30042 Label may not contain 0
30043 Label ?0 has been achieved
30045 Label cannot be empty
30055 The last change to the subgoal package state was made in a context which is no longer valid
30056 The current goal contains distinct free variables with the same names but different types, the names being ?0, and a typing context is being maintained. These free variables have not been put in the typing context
30059 The current goal contains two or more distinct free variables with the same name but different types, the name being ?0, and a typing context is being maintained. These free variables have not been put in the typing context
30061 The tactic generated an invalid proof (?0). The goal state has not been changed

These messages are common to various functions in this document. Message 30055 indicates that the goal state theorem failed the *valid_thm* test: this could be a theory out of scope, a deletion of a definition, etc. Messages 30056 and 30057 are just for the user's information, though they should give cause to worry.

SML
| (∗ **pp′TS** ∗)

Description The theory will contain a constant named *pp′TS*, defined by a definition with key "pp'TS".

Definition
| $\vdash \forall\ x \bullet (pp'TS\ x) \Leftrightarrow x$

This is used in creating a term form goal. Using this constant explicitly within the subgoal package may cause unexpected behaviour.

Uses THe definition may be used when analysing goal state theorems, or using *modify_goal_state_thm* (q.v.) - both operations are only for the advanced user or extender of the system.

SML
| (∗ **subgoal_package_quiet** : *bool* ∗)

Description This is a system control, handled by *set_flag*. If set to false (the default) then the package narrates its progress as described in the design of its components. If set to true then the package will cause no output other than the actual results of functions. This includes, e.g., *print_goal* and *apply_tactic*.

Uses For running the package offline.

SML
```
(* subgoal_package_ti_context : bool *)
```

Description *subgoal_package_ti_context* is a system control flag, as handled by *set_flag*, etc. If set to true (the default) then the type context will be set and maintained, via *set_ti_context*(q.v.), to be just the free variables of the current goal, each time the current goal changes. If false, then the type context will be cleared and left unchanged by goal state changes. If the current goal has free variables with the same name and differing types this will cause *set_ti_context* to ignore those variables, raising the comment message 30056.

SML
```
(* tactic_subgoal_warning : integer control *)
```

Description Warning 30018 will be issued by *apply_tactic* (and *a*) if the tactic requests more subgoals than the number set by this control. This allows the user to avoid processing and printing large numbers of subgoals when these are probably unwanted. The default value is 20. If the value is less than zero then the warning will never be issued.

SML
```
(* undo_buffer_length : int *)
```

Description This is a system control, handled by *set_int_control*, etc, which sets the maximum number of entries that can be held on the undo buffer for each main goal: i.e. how many tactic applications, etc, may be undone. It is initially set to 12, and cannot be made negative. Any changes to this parameter will take immediate effect upon the undo buffers stored for all the main goals, i.e. if necessary they will be shortened at the point of changing the value, rather than at the point of, e.g., applying a new tactic.

SML
```
type GOAL_STATE;
```

Description This is an abstract data type that embodies a goal state, in particular it contains which goals are yet to be achieved and a theorem embedding the inference work so far. The subgoal package has a current goal state, a stack of goal states for different main goals, and a buffer of goal states to allow some operations to be undone.

See Also *print_goal_state*

SML

```
val apply_tactic : TACTIC -> unit;
val a : TACTIC -> unit;
```

Description *apply_tactic* applies a tactic to the current goal, and *a* is an alias for it. If successful, the previous goal state will be put in the undo buffer, and the new goal state, current goal, etc, will be based on the tactic's application. If the tactic returns some subgoals then the "first" of these will become the new current goal. If there is only one subgoal it will inherit the label of the previous current goal, otherwise if the old label was "label" then it will be considered in the goal state as superseded, and the new subgoals will be labeled "label.1", "label.2", etc. If it produces a theorem that achieves the current goal (i.e. the list of subgoals is empty), then the "next" goal will become the current one, and the previous goal's label will be noted as achieved.

The subgoals created, or if none, the "next" goal, will be displayed, using the format of *print_goal*(q.v), but with goal labels also given. Following the display of the new goals the subgoal package will issue warning messages about these goals if they are somehow "suspicious": for example it will warn if the goal state is not changed by applying the tactic.

Warning 30018 will be issued if the tactic requests more subgoals than the number set by control *tactic_subgoal_warning*. This allows the user to avoid processing and printing large numbers of subgoals when these are probably unwanted.

See Also *print_goal* for the display format of the goals.

Errors

```
30007  There is no current goal
30008  Result of tactic, ?0, did not match the current goal
30018  Tactic has requested ?0 subgoals, which exceeds the threshold
       set by tactic_subgoal_warning
```

SML

```
val drop_main_goal : unit -> GOAL;
```

Description Pop the current goal state from the main goal stack throwing away it and any work upon it, and making the previous entry on the stack the new current goal state, displaying the current top goal, if appropriate. The function returns the main goal dropped.

Errors

```
30010  The subgoal package is not in use
```

SML

```
val get_asm : int -> TERM;
```

Description *get_asm n* returns the *n*th assumption of the current goal.

Errors

```
30026  There is no current goal
30027  There is no assumption ?0 in the current goal
```

SML

```
val modify_goal_state_thm : (THM -> THM) -> ((string list * GOAL)list) -> unit;
```

Description *modify_goal_state_thm rule label* is a powerful hook into the subgoal package that works as follows:

1. Extract the goal state theorem
2. Apply a user-supplied inference rule *rule* to the theorem.
3. Make a new goal state, in which the goal state theorem is this new theorem.
4. In the new goal state any goals found (up to α-conversion) in the association list *label* will be labelled with their corresponding labels in the association list. Multiple entries for the same goal in the list will cause the labels to be accumulated, resulting in duplicated goals in the new goal state. If *top_goals*() (q.v.) is used for this association list then all unchanged goals will gain their original labels.
5. Label otherwise unlabelled goals with unused single natural number labels (the first available ones from the list "1", "2",...)
6. Treat this new goal state as if it had been created by a tactics application, e.g. it becomes the current goal state, the previous goal state is put on the undo list, the user is told the next goal to prove, etc.

This will issue a warning on its use should the main goal have changed, and on attempting to extract an achieved, or goal state, theorem from a goal state that is derived from the modified one. This is so that the user is warned that the result of an apparently successful *pop_thm* is not an achievement of the initially set main goal.

Uses This function is intended for system builders wishing to write extensions to the package that change the overall proof tree, not an individual goal.

Errors

```
30039 Two labels clash: ?0 and ?1
30040 Duplicate labels ?0 given for different terms
30051 Inference rule returned '?0' which is not a goal state theorem
```

SML

```
val pending_reset_subgoal_package : unit -> unit -> unit;
```

Description This function, applied to () takes a snapshot of the current subgoal package state - its stack of goal states, undo and redo buffers, and implicitly the current goal label, etc. This snapshot, if then applied to () will overwrite the then current subgoal package state with the snapshot. This does not reset, e.g., the current theory to the one at the time of taking the snapshot, so care must be taken in using this function.

Uses Primarily in saving the subgoal package state between sessions of ProofPower, via *save-_and_quit*.

SML
```
val pop_thm : unit -> THM;
```

Description If the top achieved theorem is available (i.e. the theorem whose sequent is the main goal has been achieved), this function returns it, and then pops the previous goal state (if any) off the main goal stack, restoring its current goal and labelling. If present, the new current top goal will be displayed in the format used by *print_goal*. If the current proof is incomplete the function fails, having no effect.

If the user wishes to examine the top achieved theorem without popping the main goal stack, then they should use *top_thm* (q.v.).

The user will be informed if main goal has changed from the initially set main goal, by using *modify_goal_state_thm*(q.v).

See Also *save_pop_thm*, *top_thm*

Errors
```
30010 The subgoal package is not in use
30011 The current proof is incomplete
```

SML
```
val print_current_goal : unit -> unit;
```

Description Displays, with its label, the current goal of the current goal state: the goal to which a tactic will be applied.

Errors
```
30026 There is no current goal
```

SML
```
val print_goal_state : GOAL_STATE -> unit;
```

Description Display the given goal state. This displays the main goal, the goals yet to be proven, and the current goal.

SML
```
val print_goal : GOAL -> unit;
```

Description Display a goal (i.e. a conclusion and a list of assumptions) in the manner of the other subgoal package functions. This presents the list of assumptions in the goal first, numbered by their position, and in reverse order, and then the conclusion, distinguished from the assumptions by a turnstile.

Example
```
(*      3 *)  ⌜a ⇒ ¬ b⌝
(*      2 *)  ⌜a ⇒
               a ⇒
               a ⇒ b⌝
(*      1 *)  ⌜¬ b ⇒ a⌝

(*     ?⊢ *)  ⌜a ∨ b⌝
```

where ⌜¬ $b \Rightarrow a$⌝ is the first assumption, and the second assumption is too long to fit on one line. Then with no assumptions:

Example
```
(*     ?⊢ *)  ⌜a ∨ b⌝
```

SML
```
val push_goal_state_thm : THM -> unit;
```

Description Given a theorem that is of the form of a goal state theorem (e.g. gained by *top_goal_state_thm*, q.v.), set a new current main goal to be the conclusion of the input theorem (viewed as a term form goal). The current goal in the new goal state will be the first assumption of the input theorem, viewed as a term form goal. If it is the only assumption of the theorem argument then the corresponding goal will have label ""; otherwise label "1", and the other assumptions of the theorem will become subsequent goals with labels "2","3",... This new goal state is pushed onto the main goal stack. The current undo buffer will also be stacked, and a new empty one made current.

Uses For the advanced user, interested in partial proof.

Errors
30005 ?0 cannot be viewed as a goal state theorem
30058 Two distinct variables with name ?0 occur free in the goal

SML
```
val push_goal_state : GOAL_STATE -> unit;
```

Description If the value given is "well-formed", then this function pushes the current goal state onto the main goal stack, and sets the given value as the current goal state. The most likely reason that a goal state value is ill-formed is that it is not being pushed in the same context as it was formed, e.g. it was formed in a theory that is now out of scope, e.g. because the user has changed theory since the states creation. The current undo buffer will also be stacked, and a new empty one made current.

See Also *top_goal_state*

SML
```
val push_goal : GOAL -> unit;
```

Description Sets a new current main goal, creating an appropriate goal state and pushing it onto the main goal stack. The current (and only) goal in the new goal state will be the main goal, with label "". The current goal will be displayed. The current undo buffer will also be stacked, and a new empty one made current.

See Also *set_goal*

Errors
30002 The conclusion of the goal, ?0, is not of type BOOL
30003 An assumption of the goal, ?0, is not of type BOOL
30004 Two assumption of the goal (?0 and ?1) are $\alpha-$convertible
30058 Two distinct variables with name ?0 occur free in the goal

SML
```
val redo : unit -> unit;
```

Description If the last command to affect the goal state was an *undo*(q.v) then this command will undo its effect (including leaving the undo buffer in its previous form, without mention of the *undo* or *redo*).

Errors
30014 The last command to affect the goal state was not an undo

SML
```
val save_pop_thm : string -> THM;
```

Description If the top achieved theorem is available (i.e. the theorem whose sequent is the main goal has been achieved), this function returns it, as well as saving it under the given string key on the current theory, and then pops the previous goal state (if any) of the main goal stack, restoring its current goal and labelling. If present, the new current top goal will be displayed in the format used by *print_goal*. If the current proof is incomplete, or the key is already used in the current theory, the function fails, having no effect.

The user will be informed if main goal has changed from the initially set main goal, by using *modify_goal_state_thm*(q.v).

The user will be informed if main goal has changed from the initially set main goal, by using *modify_goal_state_thm*(q.v).

See Also *pop_thm*, *top_thm*

Errors
```
30010 The subgoal package is not in use
30011 The current proof is incomplete
```

Failures also as *save_thm*, but given as originating from this function.

SML
```
val set_goal : GOAL -> unit;
```

Description This first discards, if it exists, the current main goal (but not any previously pushed main goals). It then sets a new current main goal, creating an appropriate goal state and pushing it onto the main goal stack. The current (and only) goal in the new goal state will be the main goal, with label “”. The current goal will be displayed. The current undo buffer will also be stacked, and a new empty one made current.

Defn
```
set_goal gl = (drop_main_goal() handle (Fail _) => ();
        push_goal gl);
```

Uses In restarting a proof that has “gone wrong”, perhaps by

```
set_goal(top_main_goal());
```

See Also *push_goal*

Errors
```
30002 The conclusion of the goal, ?0, is not of type BOOL
30003 An assumption of the goal, ?0, is not of type BOOL
30004 Two assumption of the goal (?0 and ?1) are α−convertible
30058 Two distinct variables with name ?0 occur free in the goal
```

SML
```
val set_labelled_goal : string -> unit;
```

Description If the string is a valid label in the current goal state, then set the corresponding goal as the current goal, and then display it.

Errors
```
30010 The subgoal package is not in use
30016 ?0 is not of the form "n1.n2....nm"
```

SML
```
val simplify_goal_state_thm : THM -> THM;
```

Description This will simplify a goal state theorem (e.g from *top_goal_state_thm*, q.v.), stripping off assumptions from the conclusion of the theorem up to the turnstile place marker, then removing the place marker itself in both conclusion and assumptions.

Uses For the advanced user, interested in partial proofs.

Errors
```
30005 ?0 cannot be viewed as a goal state theorem
```

SML
```
val subgoal_package_size : unit -> int;
```

Description This returns the size of the subgoal package's storage, in words - where one word is four bytes.

This facility is not available in all versions of ProofPower. The function will produce the following warning message and return -1 in this case

Errors
```
30060 This function is not supported in this version of ProofPower
```

SML
```
val top_current_label : unit -> string;
```

Description Returns the label of the current goal: the goal to which a tactic will be applied.

Errors
```
30026 There is no current goal
```

SML
```
val top_goals : unit -> (string list * GOAL)list;
```

Description Returns all the goals yet to be achieved, and their associated labels (they may have more than one), in the current goal state.

Uses To determine what goals are left to achieve.

Errors
```
30010 The subgoal package is not in use
```

SML
```
val top_goal_state_thm : unit -> THM;
```

Description This returns the goal state theorem of the current goal state. It is a partial proof of the main goal, though in a somewhat unwieldy form, as it encodes the main goal, and its other goals in a term form. It may be simplified by using *simplify_goal_state_thm*(q.v). The theorem is suitable for setting a new main goal, by using *push_goal_state_thm*(q.v). The user is informed if the goal state has achieved its theorem The user will also be informed if main goal has changed from the initially set main goal, by using *modify_goal_state_thm*(q.v).

Uses For the advanced user, interested in partial proofs.

Errors
```
30010 The subgoal package is not in use
```

SML
```
val top_goal_state : unit -> GOAL_STATE;
```

Description This provides the current goal state as a value: note that a goal state does not contain an undo buffer, and thus function does not return the current undo buffer.

See Also *push_goal_state*

Errors
```
30010 The subgoal package is not in use
```

SML
```
val top_goal : unit -> GOAL;
```

Description Returns the current goal of the current goal state: the goal to which a tactic will be applied.

Errors
```
30026 There is no current goal
```

SML
```
val top_labelled_goal : string -> GOAL;
```

Description Returns the goal with the given label, should it exist in the current goal state. Note that superseded and achieved goals are not available from the goal state.

Errors
```
30016 ?0 is not of the form "n1.n2....nm"
```

SML
```
val top_main_goal : unit -> GOAL;
```

Description Return the current main goal: the objective of the current proof attempt.

Errors
```
30025 There is no current main goal
```

SML
```
val top_thm : unit -> THM;
```

Description If the top achieved theorem (i.e. the theorem whose sequent is the main goal has been achieved) is available, this function returns it, without affecting the current goal state. If the current proof is incomplete the function fails.

The user will be informed if main goal has changed from the initially set main goal, by using *modify_goal_state_thm*(q.v).

See Also *pop_thm*, *save_pop_thm*

Errors
```
30010 The subgoal package is not in use
30011 The current proof is incomplete
```

SML

```
val undo : int -> unit;
```

Description *undo n* will take the *n*th entry from the undo buffer, if there are sufficient, as the current goal state. Attempting to go past the end of the buffer will cause a failure, rather than a partial undoing. A single *undo* command can itself be undone by *redo*(q.v), but otherwise entries on the undo buffer between its start and the *n*th entry will be discarded.

Note that the undo buffer is stacked on starting a new main goal (e.g. with *push_goal*), and unstacked on popping the current main goal (e.g. with *pop_thm* or *drop_main_goal*).

Errors

30010 The subgoal package is not in use
30012 Attempted to undo ?0 time?1 with only ?2 entr?3 in the undo buffer
30013 Must undo a positive number of times

8.3 General Tactics and Tacticals

SML

signature **Tactics1** = *sig*

Description This provides the first group of tactics and tacticals in ICL HOL.

SML

signature **Tactics2** = *sig*

Description This provides the second group of tactics and tacticals in ICL HOL. These are mainly concerned with the predicate calculus.

SML

signature **Tactics3** = *sig*

Description This provides a third group of tactics. They are primarily concerned with adding handling for paired abstractions.

SML

```
type GOAL       (* = SEQ *);
type PROOF      (* = THM list -> THM *);
type TACTIC     (* = GOAL -> (GOAL list * PROOF) *);
```

Description *TACTIC* is the type of tactics. The types *GOAL* and *PROOF* help to abbreviate its definition.

SML

```
type THM_TACTIC       (* = THM -> TACTIC *);

type THM_TACTICAL     (* = THM_TACTIC -> THM_TACTIC *);
```

Description These are the types of theorem tactics and theorem tacticals.

SML

val **accept_tac** : *THM -> TACTIC*;

Description Prove a goal by a theorem which is α-convertible to it.

Tactic

$$\frac{\{\ \Gamma 2\ \}\ t2}{} \qquad \begin{array}{l} accept\_tac \\ \Gamma 1 \vdash t1 \end{array}$$

where $t1$ and $t2$ are α-convertible.

Errors

9102 ?0 is not α−convertible to the goals conclusion ?1

SML

val **all_asm_ante_tac** : *TACTIC*;

Description Apply *asm_ante_tac* to every assumption in turn:

Tactic

$$\frac{\{\ t1,\ ...,\ tn\ \}\ t}{\{\}\ tn \Rightarrow ... \Rightarrow t1 \Rightarrow t} \qquad all\_asm\_ante\_tac$$

α-equivalent assumptions will only appear once in the resulting goal. Notice that the first assumption becomes the rightmost antecedent.

See Also *asm_ante_tac*, *list_asm_ante_tac*

Errors

28055 The conclusion or an assumption of goal does not have type ⌜:BOOL⌝

SML

```
val all_var_elim_asm_tac : TACTIC;
val all_var_elim_asm_tac1 : TACTIC;
val ALL_VAR_ELIM_ASM_T : (THM -> TACTIC) -> TACTIC;
val ALL_VAR_ELIM_ASM_T1 : (THM -> TACTIC) -> TACTIC;
```

Description These tactics and tacticals do variable elimination with all the appropriate assumptions of the goal. They process one or more assumptions of the form: ⌜*var* = *value*⌝ or ⌜*value* = *var*⌝, where *var* is a variable and the subterm *value* satisfies a tactic-specific requirement, eliminating the variable *var* in favour of the *value*.

If an assumption is an equation of variables, which all of the listed tactics accept, then the tactic will strip digits and the current variant suffix from the right of the two variable names, and will choose to eliminate the variable with the shortest remaining name string, taking eliminating the left hand side variable if the strings are of equal length (this is a heuristic). If the variables are the same then the assumption is just discarded with no further effect.

all_var_elim_asm_tac will first extract all the goal's assumptions, holding them in a "pool". It will examine each assumption of the required form in turn, starting at the assumptions from the head of the assumption list. To eliminate a variable *var* using an assumption it requires that the *value* to which it is equated is also a variable, or an isolated constant (this is more restrictive than *var_elim_asm_tac*). All the occurrences of the variable will be eliminated from the rest of the assumptions in the pool, and from the conclusion of the goal, and the assumption discarded from the pool. Each of the assumptions in the pool will be examined once, as the process described so far will only exceptionally introduce new equations that can be used for variable elimination.

Finally, the remaining assumptions in the pool will be returned to the goal's assumption list - if an individual assumption is unchanged then it will be returned by *check_asm_tac*, otherwise it will be stripped back into the assumption list by *strip_asm_tac*. This stripping may result in further possible variable eliminations being enabled, and indeed certain fairly unlikely combinations of assumptions and proof contexts may result in *REPEAT all_var_elim_asm_tac* not halting. *ALL_VAR_ELIM_ASM_T* allows the users choice of function to be applied to the modified assumptions, rather than *strip_asm_tac*.

all_var_elim_asm_tac1 works as *all_var_elim_asm_tac*, except that an assumption will be used to eliminate a variable *var* if the *value* to which it is equated does not contain *var* free (i.e. its requirement is as *var_elim_asm_tac*). *ALL_VAR_ELIM_ASM_T1* allows the users choice of function to be applied to the modified assumptions.

All the functions fail if they find no assumptions that can be used to eliminate variables.

Uses General purpose, and in *basic_prove_tac*.

See Also *prop_eq_prove_tac* for more sophisticated approach to these kinds of problems.

Errors

```
29028 This tactic is unable to eliminate any variable
```

SML
```
val all_β_tac : TACTIC;
```

Description This tactic will β-reduce all β-redexes in the goal's conclusion, including those redexes introduced by preceding β-reductions in the same tactic application.

Uses In most proof contexts β-reduction will be a side effect of rewriting: this tactic is intended for cases where rewriting would do "too much".

See Also *all_β_rule*, *all_β_conv*

Errors
```
27049 ?0 contains no β−redexes
```

SML
```
val all_ε_tac : TACTIC;
val ALL_ε_T : (THM -> TACTIC) -> TACTIC;
```

Description *all_ε_tac* applies *ε_tac* to all subterms of the conclusion of the goal of the form $\epsilon x \bullet t$. *ALL_ε_T* is similar but uses *ε_T* rather than *ε_tac*. The effect is to set the corresponding terms of the form $\exists x \bullet t$ as lemmas, and to derive new assumptions of the form $t[\epsilon x \bullet t/x]$.

Tactic

$$\frac{\{\ \Gamma\ \}\ t[\epsilon x_1 \bullet t_1/y_1,\ ...,\ \epsilon x_k \bullet t_k/y_k]}{\begin{array}{l}\{\ \Gamma\ \}\ \exists x_1 \bullet t_1;\ ...;\ \{\ \Gamma\ \}\ \exists x_k \bullet t_k;\\ \quad \{strip\ \ t_1[\epsilon x_1 \bullet t_1/x_1],\ ...,\\ \qquad\qquad strip\ \ t_k[\epsilon x_k \bullet t_k/x_k],\ \Gamma\ \}\ t\end{array}} \qquad \epsilon\_tac$$

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable. This occurs when the use of the choice function is in some sense irrelevant to the truth of the goal, e.g., $(\epsilon x \bullet T) = (\epsilon x \bullet T)$.

SML
```
val ante_tac : THM -> TACTIC;
```

Description Replace a goal with conclusion *t2* by *t1⇒t2*, where the antecedent, *t1*, of the implication is the conclusion of a theorem:

Tactic

$$\frac{\{\ \Gamma 2\ \}\ t2}{\{\ \Gamma 2\ \}\ t1\ \Rightarrow\ t2} \qquad ante\_tac\ (\Gamma 1\ \vdash\ t1)$$

where the assumptions, $\Gamma 1$, of the theorem are contained in the assumptions, $\Gamma 2$ of the goal.

Uses This is often useful if one needs to transform the conclusion of theorem e.g. by rewriting with the assumptions.

See Also *asm_tac*, *strip_asm_tac*

Errors
```
28027 Conclusion of goal does not have type ⌜:BOOL⌝
```

SML
```
val asm_ante_tac : TERM -> TACTIC ;
```

Description Bring a term out of the assumption list into the goal as the antecedent of an implication.

Rule

$$\frac{\{\ \Gamma,\ t1'\ \}\ t2}{\{\ \Gamma\ \} \vdash t1 \Rightarrow t2} \qquad \begin{array}{l} asm\_ante\_tac \\ \ulcorner t1 \urcorner \end{array}$$

where *t1* and *t1′* are α-convertible. Note that all assumptions α-convertible with *t1* are removed.

Uses Typically to make the assumption amenable to manipulation, e.g. by a rewriting tactic.

See Also *list_asm_ante_tac*, *all_asm_ante_tac*, *swap_asm_concl_tac*, *DROP_ASM_T*.

Errors

28052 Term ?0 is not in the assumptions
28055 The conclusion or an assumption of goal does not have type ⌜:BOOL⌝

SML
```
val asm_tac : THM -> TACTIC;
```

Description *asm_tac thm* is a tactic which adds the conclusion of the theorem, *thm*, into the assumptions of a goal:

Tactic

$$\frac{\{\ \Gamma 2\ \}\ t2}{\{\ t1,\ \Gamma 2\ \}\ t2} \qquad \begin{array}{l} asm\_tac \\ \Gamma 1 \vdash t1 \end{array}$$

SML

```
val back_chain_tac : THM list -> TACTIC;
val bc_tac : THM list -> TACTIC;
```

Description *back_chain_tac* is a tactic which uses theorems whose conclusions are possibly universally quantified implications or bi-implications, to reason backwards from the conclusion of a goal. (*bc_tac* is an alias for *back_chain_tac*.) The tactic repeatedly performs the following steps:

1. Scan the list of theorems looking for an implication, $t1 \Rightarrow t2$, or a bi-implication $t1 \Leftrightarrow t2$ for which the conclusion of the goal is a substitution instance, $t2'$ say, of $t2$. If no such theorem is found then stop.
2. If in step 1, an applicable theorem, say *thm*, has been found reduce the goal to the corresponding instance of *t1* (or an existentially quantified version thereof) using *bc_thm_tac*, q.v.
3. Repeatedly apply $\forall$_*tac* or $\wedge$_*tac* until neither of these is applicable.
4. Delete *thm* from the list of theorems and return to step 1.

In step 4, only the first appearance of *thm* is removed from the list, so that one can arrange for a theorem to be used more than once by the tactic by putting several copies of it in the list.

For example:

Example

$$\frac{\{\ \Gamma\ \}\ t3'}{\{\ \Gamma\ \}\ t4';\ \{\ \Gamma\ \}\ t5'} \qquad \begin{array}{l} \Rightarrow\_tac \\ [\Gamma 1 \vdash t1 \wedge (\forall x \bullet t2) \Rightarrow t3, \\ \Gamma 2 \vdash t4 \Leftrightarrow t1, \\ \Gamma 3 \vdash t5 \Rightarrow t2, \end{array}$$

(Here $t3'$ is some substitution instance of $t3$ and $t4'$ and $t5'$ are the corresponding instances of $t4$ and $t5$.)

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

See Also *bc_thm_tac* (which is used to perform step 2).

Errors

29012 Theorem ?0 is not of the form ‘Γ ⊢ ∀ x1 ... xn • u ⇔ v‘
or ‘Γ ⊢ ∀ x1 ... xn • u ⇒ v‘

SML
```
val back_chain_thm_tac : THM -> TACTIC;
val bc_thm_tac : THM -> TACTIC;
```

Description *back_chain_thm_tac* is a tactic which uses a theorem whose conclusion is a possibly universally quantified implication or bi-implication to chain backwards one step from the conclusion of a goal. (*bc_thm_tac* is an alias for *back_chain_thm_tac*.) The effect is as follows:

Tactic

$$\frac{\{\ \Gamma\ \}\ t2'}{\{\ \Gamma\ \}\ t1'} \qquad \begin{array}{l} bc\_thm\_tac \\ \Gamma 1 \vdash t1 \Rightarrow t2 \end{array}$$

Tactic

$$\frac{\{\ \Gamma\ \}\ t2'}{\{\ \Gamma\ \}\ t1'} \qquad \begin{array}{l} bc\_thm\_tac \\ \Gamma 1 \vdash t1 \Leftrightarrow t2 \end{array}$$

where $t2'$ is an instance (under type instantiation and substitution) of $t2$ and $t1'$ is the corresponding instance of $t1$. If $t1'$ contains free variables which do not appear in the assumptions of the instantiated theorem or in $t2'$, then the new subgoal $t1'$ will be existentially quantified over these variables. For example,

Example

$$\frac{\{\ \Gamma\ \}\ a < b}{\{\ \Gamma\ \}\ \exists\ i \bullet\ a < i \land i < b} \qquad \begin{array}{l} bc\_thm\_tac \\ \vdash \forall\ m\ i\ n \bullet\ m < i \land i < n \Rightarrow m < n \end{array}$$

Note that, bi-implications are in effect treated as right-to-left rewrite rules at the top level by this tactic. The standard rewriting mechanisms may be used for left-to-right rewriting.

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

See Also *back_chain_tac* (which supplies a more general facility).

Errors

29011 Conclusion of the goal is not an instance of: ?0
29012 Theorem ?0 is not of the form ‘Γ ⊢ ∀ x1 ... xn • u ⇔ v‘
or ‘Γ ⊢ ∀ x1 ... xn • u ⇒ v‘

SML
```
val bad_proof : string -> 'a
```

Description *bad_proof name* is equivalent to *error name 9001* []. *bad_proof* is for use in low level tactical programming to report the error situation when the proof generated by a tactic is supplied with the wrong number of arguments. (This will not happen for the usual use of tactics with *tac_proof* or within the subgoal package):

Errors

9001 the proof of the subgoals has produced the wrong number of theorems

Uses Specialised low-level tactic programming.

SML
```
val CASES_T2 : TERM -> (THM -> TACTIC) ->
        (THM -> TACTIC) -> TACTIC;
```

Description Do a case split on a given boolean term using two tactic generating functions:

$CASES\_T2\ t1\ ttac1\ ttac2\ (\{\Gamma\}\ t2) = ttac1(t1 \vdash t1)(\{\Gamma\}\ t2)\ ;\ ttac2(\neg t1 \vdash \neg t1)(\{\Gamma\}\ t2)$

See Also *cases_tac*, $\vee$_*THEN*, *CASES_T*

Errors
28022 ?0 is not boolean

SML
```
val cases_tac : TERM -> TACTIC;
```

Description Do a case split on a given boolean term.

Tactic

$$\frac{\{\ \Gamma\ \}\ t2}{\{strip\ t1,\ \Gamma\ \}\ t2;\ \{strip\ \neg t1,\ \Gamma\ \}\ t2} \qquad \begin{array}{l} cases\_tac \\ \ulcorner t1 \urcorner \end{array}$$

See Also *CASES_T*, $\vee$_*THEN*

Errors
28022 ?0 is not boolean

SML
```
val CASES_T : TERM -> (THM -> TACTIC) -> TACTIC;
```

Description Do a case split on a given boolean term using a tactic generating function:

$CASES\_T\ t1\ ttac\ (\{\Gamma\}\ t2) = ttac(t1 \vdash t1)(\{\Gamma\}\ t2)\ ;\ ttac(\neg t1 \vdash \neg t1)(\{\Gamma\}\ t2)$

See Also *cases_tac*, $\vee$_*THEN*, *CASES_T2*

Errors
28022 ?0 is not boolean

SML
```
val CHANGED_T : TACTIC -> TACTIC;
```

Description *CHANGED_T tac* is a tactic which applies *tac* to the goal and fails if this results in a single subgoal which is $\alpha-$convertible to the original goal.

Uses *CHANGED_T* can be a useful way of ensuring termination of, e.g., rewriting tactics.

Errors
9601 the tactic did not change the goal

SML
```
val check_asm_tac : THM -> TACTIC;
```

Description *check_asm_tac thm* is a tactic which checks the form of the theorem, *thm*, and then takes the first applicable action from the following table:

thm	action
$\Gamma \vdash t$	proves goal if its conclusion is t
$\Gamma \vdash T$	as *id_tac* (i.e. the theorem is discarded)
$\Gamma \vdash F$	proves goal
$\Gamma \vdash \neg t$	proves goal if t in assumptions, else as *asm_tac*
$\Gamma \vdash t$	proves goal if $\neg t$ in assumptions, else as *asm_tac*

During the search through the assumptions in the last two cases, *check_asm_tac* also checks to see whether any of the assumptions is equal to the conclusion of the goal, and if so proves the goal. It also checks to see if the conclusion of the theorem is already an assumption, in which case the tactic has no effect. When all the assumptions have been examined, if none of the above actions is applicable, the conclusion of the theorem is added to the assumption list.

Uses Tactic programming.

See Also *strip_asm_tac*, *strip_tac*.

SML
```
val concl_in_asms_tac : TACTIC;
```

Description *concl_in_asms_tac* is a tactic which checks whether the conclusion of the goal is also in the assumptions, and if so proves the goal.

Tactic

$$\frac{\{\ \Gamma,\ t\ \}\ t'}{} \qquad concl\_in\_asms\_tac$$

where t and t' are α-convertible.

Uses Tactic programming.

See Also *strip_tac*.

Errors
```
28002 Goal does not appear in the assumptions
```

SML
```
val COND_T : (GOAL -> bool) -> TACTIC -> TACTIC -> TACTIC;
```

Description *COND_T p tac1 tac2* is a tactic which acts as *tac1* if the predicate *p* holds for the goal, otherwise it acts as *tac2*.

Example
```
COND_T (is_¬ o snd) (cases_tac ⌜X:BOOL⌝) strip_tac
```

is a tactic which does a case split on ⌜X⌝ if the goal is a negation and behaves as *strip_tac* otherwise.

Uses For constructing larger tactics, in cases where the more common idiom using *ORELSE* would not have the desired effect.

See Also *ORELSE*

Errors As determined by the arguments.

SML

val **contr_tac** : *TACTIC*;

Description A form of proof by contradiction: t holds if $\neg t \vdash F$.

(The name stands for classical contradiction, as opposed to the intuitionistic contradiction proof of *i_contr_tac*.)

Tactic

$$\frac{\{\ \Gamma\ \}\ t}{\{strip\ \neg t,\ \Gamma\}\ F} \qquad contr\_tac$$

Uses Proof by contradiction.

See Also *strip_tac*, *¬_tac*.

Errors

28027 Conclusion of goal does not have type ⌜:BOOL⌝

SML

val **CONTR_T** : (*THM* −> *TACTIC*) −> *TACTIC*;

Description A form of proof by contradiction as a tactical. *CONTR_T thmtac* is a tactic which attempts to solve a goal (Γ, t), by applying $thmtac(\neg t \vdash \neg t)$ to the goal (Γ, F).

Tactic

$$\frac{\{\ \Gamma\ \}\ t}{thmtac\ (\neg t\ \vdash\ \neg t)\ (\{\Gamma\}\ F)} \qquad CONTR\_T\ thmtac$$

Uses Proof by contradiction in combination with a theorem tactic.

See Also *contr_tac*, *¬_T*.

Errors

28027 Conclusion of goal does not have type ⌜:BOOL⌝

SML

val **conv_tac** : *CONV* −> *TACTIC*;

Description *conv_tac conv* is a tactic which applies the conversion *conv* to the conclusion of a goal, and replaces the conclusion of the goal with the right-hand side of the resulting equational theorem if this is successful:

Tactic

$$\frac{\{\ \Gamma 2\ \}\ t2}{\{\ \Gamma 2\ \}\ t1} \qquad asm\_tac\ conv$$

where $conv\ t2 = (\Gamma 1 \vdash t2 = t1)$.

Errors

9400 the conversion returned ‘?0‘ which is not of the form:
‘... ⊢ ?1 ⇔ ...‘

SML
```
val CONV_THEN : CONV -> THM_TACTICAL;
```

Description *CONV_THEN conv thmtac* is a theorem tactic which first uses *conv* to transform the conclusion of a theorem and then acts as *thmtac*.

```
(CONV_THEN) conv thmtac thm = thmtac (conv thm)
```

Uses For use in programming theorem tacticals. The function may be partially evaluated with only its conversion, theorem tactic and theorem arguments.

Errors
```
9400  the conversion returned '?0' which is not of the form:
      '... ⊢ ?1 ⇔ ...'
```

SML
```
val discard_tac : 'a -> TACTIC;
val k_id_tac : 'a -> TACTIC;
```

Description A tactic that discards its argument, but otherwise has no effect. *k_id_tac* is an alias for *discard_tac*.

Uses Can be used to remove unwanted assumptions : *a (POP_ASM_T discard_tac)* discards the top-most assumption. This usage of *discard_tac* may strengthen the goal. ie it may result in unprovable subgoals even when the original goal was provable.

SML
```
val DROP_ASMS_T : (THM list -> TACTIC) -> TACTIC;
```

Description *DROP_ASMS_T thmstac* is a tactic which applies *asm_rule* to each assumption of the subgoal, giving a list of theorems, *thms* say, then removes all the assumptions of the goal and then acts as *thmstac thms*.

Tactic

$$\frac{\{\ \Gamma\ \}\ t}{thmstac\ (map\ asm\_rule\ \Gamma)\ (\{\}\ t)} \qquad \begin{array}{l} DROP\_ASMS\_T \\ thmstac \end{array}$$

Uses To use all the assumptions as theorems.

Errors As for *thmstac*.

SML
```
val DROP_ASM_T : TERM -> (THM -> TACTIC) -> TACTIC;
```

Description *DROP_ASM_T asm thmtac* is a tactic which removes *asm* from the assumption list and then acts as *thmtac*($asm \vdash asm$).

Tactic

$$\frac{\{\ \Gamma,\ asm'\ \}\ t}{thmtac\ (asm \vdash asm)\ (\{\ \Gamma\ \}\ t)} \qquad \begin{array}{l} DROP\_ASM\_T \\ \ulcorner asm \urcorner \\ thmtac \end{array}$$

where *asm* and asm' are α-convertible.

Uses To use an assumption as a theorem

Errors
```
9301  the term ?0 is not in the assumption list
```

SML
```
val DROP_FILTER_ASMS_T : (TERM -> bool) ->
        (THM list -> TACTIC) -> TACTIC;
```

Description *DROP_FILTER_ASMS_T pred thmstac* is a tactic which applies *asm_rule* to each assumption of the subgoal that satisfies *pred*, giving a list of theorems, *thms* say, then removes all the selected assumptions of the goal and then acts as *thmstac thms*.

Tactic

$$\frac{\{\ \Gamma\ \}\ t}{\begin{array}{c} thmstac\ (map\ asm\_rule\ (\Gamma \cap pred)) \\ (\{\Gamma \setminus pred\}\ t) \end{array}} \quad \begin{array}{l} DROP\_FILTER\_ASMS\_T \\ pred \\ thmstac \end{array}$$

Uses To use all the selected assumptions as theorems.

Errors As for *thmstac*.

SML
```
val DROP_NTH_ASM_T : int -> (THM -> TACTIC) -> TACTIC;
```

Description *DROP_NTH_ASM_T i thmtac* is a tactic which applies *asm_rule* to the *i*-th assumption of the goal, giving a theorem, *thm* say, and then removes *asm* from the assumptions and acts as *thmtac thm*.

Assumptions are numbered *1, 2 . . .*, so that, e.g., *DROP_NTH_ASM_T 1* is the same as *POP_ASM_T*

Tactic

$$\frac{\{\ a1,\ ...,\ an\ \}\ t}{thmtac\ (asm\_rule\ [ai])\ (\{\ \Gamma \setminus ai\}\ t)} \quad \begin{array}{l} DROP\_NTH\_ASM\_T \\ i \\ thmtac \end{array}$$

Uses To use an assumption as a theorem, treating the assumption list as an array.

Errors
```
9303   the index ?0 is out of range
```

SML

val **eq_sym_asm_tac** : *TERM −> TACTIC*;
val **eq_sym_nth_asm_tac** : *int −> TACTIC*;

Description THese two tactics identify an assumption (either by being equal to the term argument, or by index number). They take it from the assumption list, use symmetry upon it to reverse any equations (or bi-implications) (though equations embedded within other equations will not be reversed), and then strip the result into the assumption list. The tactics fail if there are no equations to reverse.

Tactic

$$\frac{\{\Gamma 1,\ t[x = y,\ p = q,...],\ \Gamma 2\}\ cnc}{\{strip\ t[y = x,\ q = p,\ ...],\ \Gamma 1,\ \Gamma 2\}\ cnc} \qquad \begin{array}{l} eq\_sym\_asm\_tac \\ t[x = y,\ p = q,...] \end{array}$$

Tactic

$$\frac{\begin{array}{c}\{t1,...,tn{-}1,\ tn[x = y,\ p = q,...],\\ tn{+}1,...\}\ cnc\end{array}}{\begin{array}{c}\{strip\ tn[y = x,\ q = p,\ ...],\\ t1,...,tn{-}1,\ tn{+}1,...\}\ cnc\end{array}} \qquad \begin{array}{l} eq\_sym\_nth\_asm\_tac \\ n \end{array}$$

Definition

```
fun eq_sym_asm_tac asm = DROP_ASM_T asm
        (strip_asm_tac o conv_rule(ONCE_MAP_C eq_sym_conv));
fun eq_sym_nth_asm_tac n = DROP_NTH_ASM_T n
        (strip_asm_tac o conv_rule(ONCE_MAP_C eq_sym_conv));
```

Example

Assumption	Becomes
⌜x = y⌝	⌜y = x⌝
⌜∀ x y • x ⇔ y⌝	⌜∀ x y • y ⇔ x⌝
⌜f(x = (p = q))⌝	⌜f((p = q) = x)⌝
⌜x = y ∧ p = q⌝	⌜x = y⌝, ⌜p = q⌝

Errors

9301	*the term ?0 is not in the assumption list*
9303	*the index ?0 is out of range*
28053	*?0 contains no equations*

SML

val **EVERY_TTCL** : *THM_TACTICAL list −> THM_TACTICAL*;

Description *EVERY_TTCL* is a theorem tactical combinator.

EVERY_TTCL [*ttcl1*, *ttcl2*, ...] = *ttcl1 THEN_TTCL ttcl2 THEN_TTCL* ...

EVERY_TTCL [] acts as *ID_THEN*.

Uses For use in programming theorem tacticals.

SML
```
val EVERY_T : TACTIC list -> TACTIC;
val EVERY : TACTIC list -> TACTIC;
```

Description *EVERY_T tlist* is a tactic that applies the head of *tlist* to its subgoal, and recursively applies the tail of *tlist* to each resulting subgoal. *EVERY* is an alias for *EVERY_T*. *EVERY* [] is equal to *id_tac*.

Example

```
EVERY [∀_tac, ∧_tac, ∀_tac]
        is equivalent to
∀_tac THEN ∧_tac THEN ∀_tac
```

Errors As for the tactics in the list.

SML
```
val fail_tac : TACTIC;
```

Description A tactic that always fails. This is the identity for the tactical *ORELSE_T*

Uses For constructing larger tactics.

Errors
```
9201    failed as requested
```

SML
```
val FAIL_THEN : THM_TACTICAL;
```

Description This is a theorem tactical which always fails at the point it receives its theorem (having already been given a theorem tactic). It acts as the identity for the theorem tactical combinator *ORELSE_TTCL*.

Uses For use in programming theorem tacticals.

Errors
```
9401    failed as requested
```

SML
```
val fail_with_tac : string -> int -> (unit -> string) list -> TACTIC;
```

Description *fail_with_tac area msg inserts* is a tactic that always fails, reporting an error message via the call *fail area msg inserts*.

Uses For constructing larger tactics.

See Also *fail*

Errors As determined by the arguments.

SML
```
val FAIL_WITH_THEN : string -> int -> (unit -> string) list -> THM_TACTICAL;
```

Description *FAIL_WITH_THEN area msg inserts* is a theorem tactical that always fails when given its theorem (having already been given a theorem tactic), reporting an error message via the call *fail area msg inserts*.

Uses For constructing larger theorem tacticals.

See Also *fail*

Errors As determined by the arguments.

SML
```
val FIRST_TTCL : THM_TACTICAL list -> THM_TACTICAL;
```

Description *FIRST_TTCL* is a theorem tactical combinator. *FIRST_TTCL* [] fails on being applied to its theorem tactic and then theorem.

```
FIRST_TTCL [ttcl1, ttcl2, ...] =
        ttcl1 ORELSE_TTCL ttcl2 ORELSE_TTCL ...
```

Uses For use in programming theorem tacticals.

Errors
```
9402   the list of theorem tactics is empty
```

SML
```
val FIRST_T : TACTIC list -> TACTIC;
val FIRST : TACTIC list -> TACTIC;
```

Description *FIRST_T tlist* is a tactic that attempts to apply each tactics in *tlist* until one succeeds, or all fail. The first successful application will be the result of the tactic, and it fails if all the attempts fail. *FIRST* is an alias for *FIRST_T*. *FIRST* [] fails on being applied to any goal.

Errors
```
9105   the list of tactics is empty
```

Also as the failure of last member of a non-empty list.

SML
```
val forward_chain_tac : THM list -> TACTIC;
val fc_tac : THM list -> TACTIC;
val all_forward_chain_tac : THM list -> TACTIC;
val all_fc_tac : THM list -> TACTIC;
val asm_forward_chain_tac : THM list -> TACTIC;
val asm_fc_tac : THM list -> TACTIC;
val all_asm_forward_chain_tac : THM list -> TACTIC;
val all_asm_fc_tac : THM list -> TACTIC;
```

Description These are tactics which use theorems whose conclusions are implications, or from which implications can be derived using the canonicalisation function *fc_canon*, q.v., to reason forwards from the assumptions of a goal. (The names with *fc* are aliases for the corresponding ones with *forward_chain*.)

The basic step is to take a theorem of the form $\Gamma \vdash t1 \Rightarrow t2$ and an assumption of the form $t1'$ where $t1'$ is a substitution instance of $t1$ and to deduce the corresponding instance of $t2'$. The new theorem, $\Delta \vdash t2'$ say, may then be stripped into the assumptions.

In the case of *fc_tac* the implicative theorem is always derived from the list of theorems given as an argument. In the case of *asm_fc_tac* the assumptions are also used. In all of the tactics the rule *fc_canon* is used to derive an implicative canonical form from the candidate implicative theorems. Normally combination of an implicative theorem and an assumption is then tried in turn and all resulting theorems are stripped into the assumptions of the goal. However, if the chaining results contain a theorem whose conclusion is $\ulcorner F \urcorner$ then the first such found will be stripped into the assumptions, and all other theorems discarded.

If one of the implications has the form $t1 \Rightarrow t2 \Rightarrow t3$ or $t1 \wedge t2 \Rightarrow t3$ and if assumptions matching $t1$ and $t2$ are available, *fc_tac* or *asm_fc_tac* will derive an intermediate implication $t2 \Rightarrow t3$ and *asm_fc_tac* could then be used to derive $t3$. The variants with *all_* may be used to derive $t3$ directly without generating any intermediate implications in the assumptions. They work like the corresponding tactic without *all_* but any theorems which are derived which are themselves implications are not stripped into the assumptions but instead are used recursively to derive further theorems. When no new implications are derivable all of the non-implicative theorems derived during the process are stripped into the assumptions.

Note that the use of *fc_canon* implies that conversions from the proof context are applied to generate implications. E.g., in an appropriate proof-context covering set theory, $a \subseteq b$ might be treated as the implication $\forall x \bullet x \in a \Rightarrow x \in b$. Also variables which appear free in a theorem are not considered as candidates for instantiation (in order to give some control over the number of results generated). The tacticals, *FC_T1* and *ASM_FC_T1* may be used to avoid the use of *fc_canon*.

For example, the tactic:

```
asm_fc_tac[] THEN asm_fc_tac[]
```

will prove the goal:

$$\{p\ x,\ \forall x \bullet p\ x \Rightarrow q\ x,\ \forall x \bullet q\ x \Rightarrow r\ x\}\ r\ x.$$

See Also *bc_tac*, *FC_T*, *ASM_FC_T*, *FC_T1*, *ASM_FC_T1*.

SML

```
val FORWARD_CHAIN_T :
        (THM list -> TACTIC) -> THM list -> TACTIC;
val FC_T :
        (THM list -> TACTIC) -> THM list -> TACTIC;
val ALL_FORWARD_CHAIN_T :
        (THM list -> TACTIC) -> THM list -> TACTIC;
val ALL_FC_T :
        (THM list -> TACTIC) -> THM list -> TACTIC;
val ASM_FORWARD_CHAIN_T :
        (THM list -> TACTIC) -> THM list -> TACTIC;
val ASM_FC_T :
        (THM list -> TACTIC) -> THM list -> TACTIC;
val ALL_ASM_FORWARD_CHAIN_T :
        (THM list -> TACTIC) -> THM list -> TACTIC;
val ALL_ASM_FC_T :
        (THM list -> TACTIC) -> THM list -> TACTIC;
```

Description These are tacticals which use theorems whose conclusions are implications, or from which implications can be derived, to reason forwards from the assumptions of a goal. (The tacticals with *FC* are aliases for the corresponding ones with *FORWARD_CHAIN*.)

The description of *fc_tac* should be consulted for the basic forward chaining algorithms used. The significance of the final argument and of the presence or absence of *ASM* and *ALL* in the name is exactly as for *fc_tac* and its relatives.

The tacticals allow variation of the tactic generating function used to process the theorems derived by the forward inference. The tactic generating function to be used is given as the first argument.

Examples *fc_tac* is the same as: *FC_T (MAP_EVERY strip_asm_tac)*.

To rewrite the goal with the results of the forward inference one could use *FC_T rewrite_tac*.

See Also *fc_tac*, *asm_fc_tac*, *bc_tac*, *FC_T1*.

SML
```
val FORWARD_CHAIN_T1 :
    (THM -> THM list) -> (THM list -> TACTIC) -> THM list -> TACTIC;
val FC_T1 :
    (THM -> THM list) -> (THM list -> TACTIC) -> THM list -> TACTIC;
val ALL_FORWARD_CHAIN_T1 :
    (THM -> THM list) -> (THM list -> TACTIC) -> THM list -> TACTIC;
val ALL_FC_T1 :
    (THM -> THM list) -> (THM list -> TACTIC) -> THM list -> TACTIC;
val ASM_FORWARD_CHAIN_T1 :
    (THM -> THM list) -> (THM list -> TACTIC) -> THM list -> TACTIC;
val ASM_FC_T1 :
    (THM -> THM list) -> (THM list -> TACTIC) -> THM list -> TACTIC;
val ALL_ASM_FORWARD_CHAIN_T1 :
    (THM -> THM list) -> (THM list -> TACTIC) -> THM list -> TACTIC;
val ALL_ASM_FC_T1 :
    (THM -> THM list) -> (THM list -> TACTIC) -> THM list -> TACTIC;
```

Description These are tacticals which use theorems whose conclusions are implications, or from which implications can be derived, to reason forwards from the assumptions of a goal. (The tacticals with *FC* are aliases for the corresponding ones with *FORWARD_CHAIN*.)

The description of *fc_tac* should be consulted for the basic forward chaining algorithms used. The significance of the final argument and of the presence or absence of *ASM* and *ALL* in the name is exactly as for *fc_tac* and its relatives.

The tacticals allow variation of the canonicalisation function used to obtain implications from the argument theorems and of the tactic generating function used to process the theorems derived by the forward inference. The canonicalisation function to use is the first argument and the tactic generating function is the second. (Related tacticals with names ending in *T* rather than *T1* are also available for the simpler case when wants to use the same canonicalisation function as *fc_tac* and just to vary the tactic generating function.)

Examples If the theorem argument comprises only implications which are to be used without canonicalisation, one might use: *FC_T1 id_canon (MAP_EVERY strip_asm_tac)*.

If one has an instance of *t1* as an assumption and one wishes to use the bi-implication in a theorem of the form $\vdash t1 \Rightarrow (t2 \Leftrightarrow t3)$ for rewriting, one might use *FC_T1 id_canon rewrite_tac*.

See Also *fc_tac*, *asm_fc_tac*, *bc_tac*, *FC_T*.

SML
```
val f_thm_tac : THM -> TACTIC;
```

Description Prove a goal by using a theorem of the form $\Gamma \vdash F$.

Tactic

$$\frac{\{\ \Gamma 2\ \}\ t}{} \qquad f\_thm\_tac\ (\Gamma 1 \vdash F)$$

where the assumptions, $\Gamma 1$, of the theorem are contained in the assumptions, $\Gamma 2$, of the goal.

Errors
```
28021 ?0 does not have the form Γ ⊢ F
```

Uses In tactic programming, to use a theorem which shows that the assumptions are contradictory.

See Also *strip_asm_tac*.

SML
```
val GEN_INDUCTION_T : THM -> (THM -> TACTIC) -> TERM -> TACTIC;
val gen_induction_tac : THM -> TERM -> TACTIC;
```

Description These give general means for constructing an induction tactic from an induction principle formulated as a theorem. The term argument is the induction variable, which must be free in the conclusion of the goal to which the tactic is applied but not in the assumptions.

GEN_INDUCTION_T causes any inductive hypotheses (see below) to be passed to a tactic generating function.

gen_induction_tac thm is the same as *GEN_INDUCTION_T thm strip_asm_tac*.

The discussion below is for the tactic computed by the call *GEN_INDUCTION_T thm ttac y* applied to a goal with conclusion t.

The induction principle, *thm* has the form:

$\vdash \forall p \bullet a \Rightarrow \forall x \bullet p\ x$

E.g. the usual principle of induction for the natural numbers:

$\vdash \forall\ p \bullet\ p\ 0 \land (\forall\ n \bullet\ p\ n \Rightarrow p\ (n + 1)) \Rightarrow (\forall\ n \bullet\ p\ n)$

The induction tactic takes the following steps:

1. Use $\forall$-elimination on *thm*, (with the term $\ulcorner \lambda y \bullet t \urcorner$) and β-reduction to give an implicative theorem, $\vdash a' \Rightarrow t$ and use it to reduce the goal to a subgoal with conclusion a'.
2. Repeatedly apply $\land$_*tac* and then repeatedly apply $\forall$_*tac*.
3. To any of the resulting subgoals whose principal connective corresponds to an an implication in *thm* apply $\Rightarrow$_*T ttac*. E.g., with the usual principle of induction for the natural numbers as formulated above $\Rightarrow$_*T ttac*is applied in the inductive step but not in the base case, even if the conclusion of the goal is an implication.

The tactic also renames bound variables so that names which begin with the name of the variable in the theorem now begin with the name of the induction variable passed to the tactic.

Errors
```
29021 ?0 does not have the form '⊢ ∀p•a ⇒ ∀x•p x'
29023 The type of ?0 is not an instance of ?1
29024 ?0 is not a variable
29025 ?0 appears free in the assumptions of the goal
29026 ?0 does not appear free in the conclusion of the goal
```

SML

val **GEN_INDUCTION_T1** : THM –> (THM –> $TACTIC$) –> $TACTIC$;
val **gen_induction_tac1** : THM –> $TACTIC$;

Description These give a means for constructing an induction tactic from an induction principle formulated as a theorem, in cases where the induction variable can be inferred from the form of the theorem and the goal. They are in other respects very like *GEN_INDUCTION_T* and *gen_induction_tac thm*, q.v.

The induction theorem must be a theorem of the form:

$\vdash \forall p \bullet a \Rightarrow \forall x \bullet t[p\ x/b]$

Where t contains at least one occurrence of x. For example,

$$\vdash \forall\ p \bullet p\ \{\} \wedge (\forall\ a\ x \bullet\ a \in Finite \wedge p\ a \wedge \neg\ x \in a \Rightarrow p\ (\{x\} \cup a))$$
$$\Rightarrow (\forall\ a \bullet\ a \in Finite \Rightarrow p\ a)$$

(for which t is $a \in Finite \Rightarrow b$).

The induction tactic matches the conclusion, c, of the goal with t, uses the result to derive a theorem of the form $\vdash a' \Rightarrow c$ and then proceeds exactly like the corresponding induction tactic produced by *GEN_INDUCTION_T* and *gen_induction_tac thm* q.v.

Errors

29007 ?0 does not have the form ‘$\vdash\forall p{:}\tau\rightarrow BOOL \bullet a \Rightarrow \forall x \bullet t[p\ x/b]$‘
(where $\ulcorner x \urcorner$ must also appear in $\ulcorner t \urcorner$ other than as an argument of $\ulcorner p \urcorner$)
29009 The conclusion of the goal cannot be rewritten in the form ?0
29014 The term ?0 which matches the induction variable is not a variable

SML

val **GET_ASMS_T** : ($THM\ list$ –> $TACTIC$) –> $TACTIC$;

Description *GET_ASMS_T thmstac* is a tactic which applies *asm_rule* to each assumption of the goal, giving a list of theorems, *thms* say, and then acts as *thmstac thms*.

Tactic

$$\frac{\{\ \Gamma\ \}\ t}{\begin{array}{c} thmstac\ (map\ asm\_rule\ \Gamma) \\ (\{\ a1,...,an\ \}\ t)\end{array}} \quad \begin{array}{l} GET\_ASMS\_T \\ thmstac \end{array}$$

Uses To use all the assumptions as theorems.

Errors As for *thmstac*.

SML

val **GET_ASM_T** : $TERM$ –> (THM –> $TACTIC$) –> $TACTIC$;

Description *GET_ASM_T asm thmtac* is a tactic which checks that *asm* is in the assumption list and then acts as $thmtac(asm \vdash asm)$.

Tactic

$$\frac{\{\ \Gamma,\ asm'\ \}\ t}{thmtac\ (asm \vdash asm)\ (\{\ \Gamma,\ asm'\ \}\ t)} \quad \begin{array}{l} GET\_ASM\_T \\ \ulcorner asm \urcorner \\ thmtac \end{array}$$

where *asm* and *asm′* are α-convertible.

Uses To use an assumption as a theorem

Errors

9301 the term ?0 is not in the assumption list

SML
```
val GET_FILTER_ASMS_T : (TERM -> bool) ->
        (THM list -> TACTIC) -> TACTIC;
```

Description *GET_FILTER_ASMS_T pred thmstac* is a tactic which applies *asm_rule* to each assumption of the subgoal that satisfies *pred*, giving a list of theorems, *thms* say and then acts as *thmstac thms*.

Tactic

$$\frac{\{\ \Gamma\ \}\ t}{\begin{array}{c} thmstac\ (map\ asm\_rule\ (\Gamma \cap pred)) \\ (\{\ \Gamma\ \}\ t) \end{array}} \qquad \begin{array}{l} GET\_FILTER\_ASMS\_T \\ pred \\ thmstac \end{array}$$

Uses To use all the selected assumptions as theorems.

Errors As for *thmstac*.

SML
```
val GET_NTH_ASM_T : int -> (THM -> TACTIC) -> TACTIC;
```

Description *GET_NTH_ASM_T i thmtac* is a tactic which applies *asm_rule* to the *i*-th assumption of the goal, giving a theorem, *thm* say, and then acts as *thmtac thm*.

Assumptions are numbered *1, 2 . . .*, so that, e.g., *GET_NTH_ASM_T 1* is the same as *TOP_ASM_T*

Tactic

$$\frac{\{\ a1,\ ...,\ an\ \}\ t}{thmtac\ (asm\_rule\ [ai])\ (\{\ \Gamma\ \}\ t)} \qquad \begin{array}{l} GET\_NTH\_ASM\_T \\ i \\ thmtac \end{array}$$

Uses To use an assumption as a theorem, treating the assumption list as an array.

Errors

9303 the index ?0 is out of range

SML
```
val id_tac : TACTIC;
```

Description A tactic that always succeeds, having no effect. This is the identity for the tactical *THEN_T*.

Tactic

$$\frac{\{\ \Gamma\ \}\ t}{\{\ \Gamma\ \}\ t} \qquad id\_tac$$

Uses For constructing larger tactics.

SML
```
val ID_THEN : THM_TACTICAL;
```

Description This is the identity for the theorem tactical combinator *THEN_TTCL*.

(ID_THEN) thmtac = thmtac

Uses For use in programming theorem tacticals.

SML
```
val IF_T2 : (THM -> TACTIC) -> (THM -> TACTIC) -> TACTIC;
```

Description Reduce a conditional by applying tactic generating functions to the two cases for the selector.

Tactic

$$\frac{\{\ \Gamma\ \}\ if\ a\ then\ tt\ else\ et}{ttac1\{\ a,\ \Gamma\ \} \vdash tt;\ ttac2\{\ \neg a,\ \Gamma\ \} \vdash et} \qquad \begin{array}{l} IF\_T2 \\ ttac1\ \ ttac2 \end{array}$$

See Also $\Leftrightarrow$_T, STRIP_CONCL_T

Errors

28071 Goal is not of the form: { Γ } if a then tt else et

SML
```
val if_tac : TACTIC;
```

Description Reduce a conditional subgoal by performing a case split on the selector.

Tactic

$$\frac{\{\ \Gamma\ \}\ if\ a\ then\ tt\ else\ et}{\{strip\ a,\ \Gamma\ \}\ tt\ ;\ \ \{strip\ \neg a,\ \Gamma\ \}\ et} \qquad if\_tac$$

See Also strip_tac

Errors

28071 Goal is not of the form: { Γ } if a then tt else et

SML
```
val IF_THEN2 : (THM -> TACTIC) -> (THM -> TACTIC) ->
        (THM -> TACTIC);
```

Description A theorem tactical to apply given theorem tactics to the result of eliminating the conditional from a theorem with a conditional as its conclusion.

$IF\_THEN\ ttac\ (\Gamma \vdash if\ a\ then\ tt\ else\ et) = ttac1\ (a, \Gamma \vdash tt)\ THEN\ ttac2\ (\neg a, \Gamma \vdash et)$

The function is partially evaluated with only the theorem tactic and theorem arguments.

See Also IF_THEN, STRIP_THM_THEN

Errors

7012 ?0 is not of the form: ‘Γ ⊢ if tc then tt else te‘

SML
```
val IF_THEN : (THM -> TACTIC) -> (THM -> TACTIC);
```

Description A theorem tactical to apply a given theorem tactic to the result of eliminating the conditional from a theorem with a conditional as its conclusion.

$IF\_THEN\ ttac\ (\Gamma \vdash if\ a\ then\ tt\ else\ et) = ttac\ (\Gamma \vdash a \Rightarrow tt)\ THEN\ ttac\ (\Gamma \vdash \neg\ a \Rightarrow et)$

The function is partially evaluated with only the theorem tactic and theorem arguments.

See Also IF_THEN2, STRIP_THM_THEN

Errors

7012 ?0 is not of the form: ‘Γ ⊢ if tc then tt else te‘

SML

```
val IF_T : (THM -> TACTIC) -> TACTIC;
```

Description Reduce a conditional by applying a tactic generating function to the two cases for the selector.

Tactic

$$\frac{\{\ \Gamma\ \}\ if\ a\ then\ tt\ else\ et}{ttac\{\ a,\ \Gamma\ \} \vdash tt;\ ttac\{\ \neg a,\ \Gamma\ \} \vdash et} \qquad \begin{array}{l} IF\_T \\ ttac \end{array}$$

See Also *IF_T2*, *STRIP_CONCL_T*

Errors

28071 Goal is not of the form: { Γ } if a then tt else et

SML

```
val intro_∀_tac : (TERM * TERM) -> TACTIC;
val intro_∀_tac1 : TERM -> TACTIC;
```

Description Sometimes it is helpful to prove a goal by proving a more general conjecture has the goal as a special case. *intro_∀_tac* introduces a universal quantifier into the conclusion of a goal in order to do this.

Tactic

$$\frac{\{\ \Gamma\ \}\ t[t1/x]}{\{\ \Gamma\ \}\ \forall x \bullet t} \qquad intro\_\forall\_tac\ (t1,\ x)$$

where t is a term in which x appears free and where either $t1$ the same as x or x does not appear free in the conclusion, $t[t1/x]$, of the original goal.

Note that $t1$ need not be a variable, e.g.,

Example

$$\frac{\{\ \Gamma\ \}\ 1\ +\ 2\ >\ 0\ \Rightarrow \neg 1\ +\ 2\ =\ 0}{\{\ \Gamma\ \}\ \forall i \bullet i\ >\ 0\ \Rightarrow \neg i\ =\ 0} \qquad intro\_\forall\_tac\ (\ulcorner 1{+}2\urcorner,\ \ulcorner i{:}\mathbb{N}\urcorner)$$

intro_∀_tac1 is for use in the common case where one simply wants to take replace the goal by its universal closure over some variable. *intro_∀_tac1* $\ulcorner x \urcorner$ is equivalent to *intro_∀_tac* $(\ulcorner x\urcorner, \ulcorner x\urcorner)$.

N.B. these tactics may strengthen the goal, i.e. they may result in unprovable subgoals even when the original goal was provable.

Uses The most common use is in preparation for an inductive proof when it is necessary to generalise the conclusion in order to give stronger assumptions in the inductive step or steps.

See Also *∀_reorder_conv*

Errors

28082 ?0 does not appear free in the goal
28083 ?0 appears free in the goal and is not the same as ?1

SML

```
val i_contr_tac : TACTIC;
```

Description Prove a goal by showing that the assumptions are contradictory, in an intuitionistic manner.

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

Tactic

$$\frac{\{\ \Gamma\ \}\ t}{\{\ \Gamma\ \}\ F} \qquad i\_contr\_tac$$

Uses If a proof is to be carried out by showing the assumptions inconsistent, then the conclusion of the subgoal is irrelevant and may be removed.

SML

```
val lemma_tac : TERM -> TACTIC;
```

Description Introduce a lemma (the term argument) to be proved, and then added as an assumption.

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

Tactic

$$\frac{\{\ \Gamma\ \}\ t2}{\{\ \Gamma\ \}\ t1;\quad \{strip\ t1,\ \Gamma\}\ t2} \qquad lemma\_tac\ \ulcorner t1 \urcorner$$

See Also *LEMMA_T*

Errors

9603 the term ?0 is not boolean

SML

```
val LEMMA_T : TERM -> (THM -> TACTIC) -> TACTIC;
```

Description *LEMMA_T newsg thmtac* is a tactic which sets *newsg* as a new subgoal and applies *thmtac*(*newsg*⊢*newsg*) to the original goal.

Tactic

$$\frac{\{\ \Gamma\ \}\ t}{\{\ \Gamma\ \}\ newsg;\quad thmtac(newsg \vdash newsg)\ (\{\ \Gamma\ \}\ t)} \qquad LEMMA\_T\ newsg\ thmtac$$

Uses For use in tactic programming and in interactive use where *lemma_tac* is not appropriate.

Errors

9603 the term ?0 is not boolean

See Also *lemma_tac*.

SML

```
val list_asm_ante_tac : TERM list -> TACTIC;
```

Description Repeatedly apply *asm_ante_tac*.

Rule

$$\frac{\{\ \Gamma,\ t1,\ ...,\ tn\ \}\ t}{\{\ \Gamma\ \}\ t1 \Rightarrow ... \Rightarrow tn \Rightarrow t} \qquad \begin{array}{l} list\_asm\_ante\_tac \\ [\ulcorner t1 \urcorner,...,\ulcorner tn \urcorner] \end{array}$$

α-equivalent assumptions will only appear once in the resulting goal, in their rightmost position, (which also means that duplicates in the list are ignored).

See Also *asm_ante_tac*, *all_asm_ante_tac*

Errors

28052 Term ?0 is not in the assumptions
28055 The conclusion or an assumption of goal does not have type ⌜:BOOL⌝

SML

```
val LIST_DROP_ASM_T : TERM list -> (THM list -> TACTIC) -> TACTIC;
```

Description *LIST_DROP_ASM_T* [*asm_1*, *asm_2*, ...] *thmtac* is a tactic which removes the *asm_1*, *asm_2*, ... from the assumption list and then acts as

thmtac[(*asm_1*⊢*asm_1*), (*asm_2*⊢*asm_2*), ...]

Tactic

$$\frac{\{\ \Gamma,\ asm1',...\ \}\ t}{thmtac\ [(asm1\ \vdash\ asm1),...]\ (\{\ \Gamma\ \}\ t)} \qquad \begin{array}{l} LIST\_DROP\_ASM\_T \\ [\ulcorner asm1 \urcorner,...] \\ thmtac \end{array}$$

where *asm_i* and *asm_i′* are α-convertible.

Uses To use assumptions as theorems

Errors

9301 the term ?0 is not in the assumption list

SML

```
val LIST_DROP_NTH_ASM_T : int list ->
        (THM list -> TACTIC) -> TACTIC;
```

Description *LIST_DROP_NTH_ASM_T* [*i*, *j*, ...] *thmtac* is a tactic which applies *asm_rule* to the *i*-th, *j*-th, etc assumptions of the goal, giving theorems, *thm_i*, *thm_j*, etc, say, and then removes the *asm_i*, *asm_j* from the assumptions and acts as *thmtac* [*thm_i*, *thm_j*, ...].

Tactic

$$\frac{\{\ a1,\ ...,\ an\ \}\ t}{\begin{array}{c} thmtac\ [(asm\_rule\ [ai]), \\ (asm\_rule\ [aj]),...] \\ (\{\ \Gamma \setminus \{ai,\ aj,\ ...\}\}\ t) \end{array}} \qquad \begin{array}{l} DROP\_NTH\_ASM\_T \\ [i,j,...] \\ thmtac \end{array}$$

Uses To use assumptions as theorems, treating the assumption list as an array.

Errors

9303 the index ?0 is out of range

SML
```
val LIST_GET_ASM_T : TERM list -> (THM list -> TACTIC) -> TACTIC;
```

Description *LIST_GET_ASM_T* [*asm_1*, *asm_2*, ...] *thmtac* is a tactic which checks that all the *asm_1*, *asm_2*, ... are in the assumption list and then acts as

$$thmtac[(asm\_1 \vdash asm\_1), (asm\_2 \vdash asm\_2), ...]$$

Tactic

$$\frac{\{\ \Gamma,\ asm1',\ ...\ \}\ t}{\begin{array}{c} thmtac\ [(asm1\ \vdash\ asm1),...] \\ (\{\ \Gamma,\ asm',...\ \}\ t)\end{array}} \qquad \begin{array}{l} LIST\_GET\_ASM\_T \\ [\ulcorner asm1 \urcorner,...] \\ thmtac \end{array}$$

where *asm_i* and *asm_i′* are α-convertible.

Uses To use a list of assumptions as theorems

Errors

9301 the term ?0 is not in the assumption list

SML
```
val LIST_GET_NTH_ASM_T : int list -> (THM list -> TACTIC) -> TACTIC;
```

Description *LIST_GET_NTH_ASM_T* [*i*, *j*, ...] *thmtac* is a tactic which applies *asm_rule* to the *i*-th, *j*-th, etc, assumption of the goal, giving theorems, *thm_i*, *thm_j*, etc, say, and then acts as *thmtac* [*thm_i*, *thm_j*, ...].

Tactic

$$\frac{\{\ a1,\ ...,\ an\ \}\ t}{thmtac\ [(asm\_rule\ [ai]),...]\ (\{\ \Gamma\ \}\ t)} \qquad \begin{array}{l} LIST\_GET\_NTH\_ASM\_T \\ [i,...] \\ thmtac \end{array}$$

Uses To use assumptions as theorems, treating the assumption list as an array.

Errors

9303 the index ?0 is out of range

SML
```
val list_simple_∃_tac : TERM list -> TACTIC ;
```

Description Provide a list of witnesses for an interated existential subgoal.

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

Tactic

$$\frac{\{\ \Gamma\ \}\ \exists\ x1,x2...\ \bullet\ t2[x1,x2,...]}{\{\ \Gamma\ \}\ t2[t1',t2'...]} \qquad \begin{array}{l} list\_simple\_\exists\_tac \\ [\ulcorner t1 \urcorner,\ulcorner t2 \urcorner,...] \end{array}$$

where $t1', t2', \ldots$ are $t1, t2, \ldots$, type instantiated to have the same type as $x1, x2, \ldots$.

See Also *simple_∃_tac*

Errors

29008 Cannot match witness ?0 to variable ?1
29015 The list of witnesses is longer than the list of existentially quantified variables in ?0
29016 The list of witnesses is empty
29017 Goal is not of the form: { Γ } ∃ x • t2[x]

SML

```
val list_swap_asm_concl_tac : TERM list -> TACTIC;
val list_swap_nth_asm_concl_tac : int list -> TACTIC;
```

Description Strip the negation of current goal into the assumption list and make some assumptions, suitably negated, into a disjunction forming the current goal. If the list is empty then the conclusion will become ⌜F⌝.

Tactic

$$\frac{\{\ \Gamma\ \}\ t2}{\{strip\ ⌜\neg\ t2⌝,\ \Gamma - \{⌜t1⌝,...,⌜tn⌝\}\}\ \neg\ t1 \lor ... \lor \neg\ tn}$$
list_swap_asm_concl_tac [⌜$t1$⌝,...,⌜tn⌝]

Tactic

$$\frac{\{\ \Gamma\ \}\ t}{\{strip\ ⌜\neg\ t⌝,\ \Gamma - \{⌜tp⌝,\ ...,\ ⌜tq⌝\}\}\ \neg\ tp \lor ...\ \neg\ tq}$$
list_swap_nth_asm_concl_tac [p,...,q]

If any assumption is a negated term then the double negation will be eliminated.

See Also Other *swap* and *SWAP* functions.

Errors

9303 the index ?0 is out of range
28052 Term ?0 is not in the assumptions

SML

```
val LIST_SWAP_ASM_CONCL_T
        : TERM list -> (THM -> TACTIC) -> TACTIC;
val LIST_SWAP_NTH_ASM_CONCL_T
        : int list -> (THM -> TACTIC) -> TACTIC;
```

Description Process the negation of current goal with the supplied theorem tactic and make some assumptions, suitably negated, into a disjunction forming the current goal.

Tactic

$$\frac{\{\ \Gamma\ \}\ t}{ttac(asm\_rule\ ⌜\neg t⌝\ )\ (\{\Gamma - \{⌜tp⌝,...⌜tq⌝\}\}\ \neg\ t1)}$$
LIST_SWAP_ASM_CONCL_T [⌜tp⌝,...⌜tq⌝] *ttac*

Tactic

$$\frac{\{\ \Gamma\ \}\ t}{ttac(asm\_rule\ ⌜\neg t⌝\ )\ (\{\Gamma - \{⌜tp⌝,\ ...,\ ⌜tq⌝\}\}\ \neg\ tm)}$$
LIST_SWAP_NTH_ASM_CONCL_T [p, ..., q] *ttac*

If an assumption is a negated term then the double negation will be eliminated. If the list is empty then the conclusion (before applying the tactic argument) will become ⌜F⌝.

See Also Other *swap* and *SWAP* functions.

Errors

9303 the index ?0 is out of range
28052 Term ?0 is not in the assumptions
28027 Conclusion of goal does not have type ⌜:BOOL⌝

SML
```
val MAP_EVERY_T : ( 'a -> TACTIC) -> 'a list -> TACTIC;
val MAP_EVERY : ( 'a -> TACTIC) -> 'a list -> TACTIC;
```

Description *MAP_EVERY_T mapf alist* maps *mapf* over *alist*, and then applies the resulting list of tactics to the goal in sequence (in the same manner as *EVERY*, q.v.). *MAP_EVERY* is an alias for *MAP_EVERY_T*.

Errors As the individual items generated by mapping the tactic over the list.

SML
```
val MAP_FIRST_T : ('a -> TACTIC) -> 'a list -> TACTIC;
val MAP_FIRST : ('a -> TACTIC) -> 'a list -> TACTIC;
```

Description *MAP_FIRST_T mapf alist* maps *mapf* over *alist*, and then attempts to apply each resulting tactic in order, until one succeeds or all fail (in the same manner as *FIRST*, q.v.). *MAP_FIRST* is an alias for *MAP_FIRST_T*.

Errors As the last tactic.

SML
```
val map_shape : (('a list -> 'b) * int) list -> 'a list -> 'b list
```

Description *map_shape* is a means of composing functions on lists. It is intended for composing the proofs produced by tactics in tacticals such as *THEN*. Its effect is as follows:

```
        map_shape [(f1, n1), (f2, n2)... ] [a1, a2, ...]
   =    [f1[a1, ..., a(n1)], f2[a(n1+1), ..., a(n1+n2)], ...]
```

where, if there are not enough *a_i*, then unused *f_j* are ignored and the last *f_j* to be used may receive less than *n_j* elements in its argument. (This case is not expected to occur in the application of *map_shape* in tactic programming.)

Uses Specialised low-level tactic programming.

SML
```
val ORELSE_TTCL : (THM_TACTICAL * THM_TACTICAL) -> THM_TACTICAL;
```

Description *ORELSE_TTCL* is a theorem tactical combinator. It is an infix operator. (*tcl1 ORELSE_TTCL tcl2*)*th* is *tcl1 th* unless evaluation of *tcl1 th* fails, in which case it is *tcl2 th*.

Uses For use in programming theorem tacticals.

SML
```
val ORELSE_T : (TACTIC * TACTIC) -> TACTIC;
val ORELSE : (TACTIC * TACTIC) -> TACTIC;
```

Description *ORELSE_T* is a tactical used as an infix operator. *tac1 ORELSE_T tac2* is a tactic which behaves as *tac1* unless application of *tac1* fails, in which case it behaves as *tac2*. *ORELSE* is an alias for *ORELSE_T*

See Also *LIST_ORELSE_T*

Errors As the failure of *tac2*.

SML

val **pair_rw_canon** : $CANON$;

Description This is the rewrite canonicalisation function for the theory of pairs, defined as

```
val pair_rw_canon =
        REWRITE_CAN
        (REPEAT_CAN(FIRST_CAN [
        ∀_rewrite_canon,
        ∧_rewrite_canon,
        ¬_rewrite_canon,
        f_rewrite_canon,
        ⇔_t_rewrite_canon]));
```

This is the repeated application of the first applicable operation in the following list:

1. stripping universal quantifiers (paired or simple);
2. dividing conjunctive theorems into their conjuncts;
3. changing $\vdash\neg(t1\vee t2)$ to $\neg t1\wedge\neg t2$;
4. changing $\vdash\neg\exists vs\bullet t$ to $\forall vs\bullet\neg t$;
5. changing $\vdash\neg\neg t$ to $t\Leftrightarrow F$;
6. changing $\vdash\neg t$ to $t\Leftrightarrow F$;
7. if none of the above apply, changing $\vdash t$ to $\vdash t\Leftrightarrow T$.

Finally, after all this canonicalisation we then universally quantify the resulting theorems in all free variables other than those that were free in the original.

SML

val **POP_ASM_T** : $(THM\ \text{->}\ TACTIC)\ \text{->}\ TACTIC$;

Description $POP\_ASM\_T\ thmtac$ is a tactic which removes the top entry, asm say, from the assumption list and then acts as $thmtac(asm\vdash asm)$.

Tactic

$$\frac{\{\ asm,\ \Gamma\ \}\ t}{thmtac\ (asm\ \vdash\ asm)\ (\{\ \Gamma\ \}\ t)}\qquad \begin{array}{l}POP\_ASM\_T\\ \ulcorner asm\urcorner\\ thmtac\end{array}$$

Uses To use an assumption as a theorem

Errors

9302 *the assumption list is empty*

SML

```
val prim_rewrite_tac : CONV NET -> CANON -> (THM -> TERM * CONV) OPT ->
        (CONV -> CONV) -> EQN_CXT -> THM list -> TACTIC;
```

Description This is the tactic based on *prim_rewrite_conv* (q.v.), with the same parameters as that function, except for the last argument:

Tactic

$$\frac{\{\ \Gamma\ \} \vdash t}{\{\ \Gamma\ \} \vdash t'}$$

prim_rewrite_tac
(*initial_net*: *CONV NET*)
(*canon* : *CANON*)
(*epp* : (*THM -> TERM * CONV*) *OPT*)
(*traverse* : *CONV -> CONV*)
(*with_eqn_cxt* : *EQN_CXT*)
(*with_thms* : *THM list*)

where ⌜t'⌝ is the result of rewriting ⌜t⌝ in the manner prescribed by the arguments.

SML

```
val prove_tac : THM list -> TACTIC;
```

Description This tactic is an automatic proof procedure appropriate to the current proof context.

At the point of applying this tactic to its theorems it will access the current setting of proof context field *pr_tac*, apply it to the theorem list immediately, and then to the goal, with its assumptions temporarily removed when available (i.e. the result is partially evaluated with only the list of theorems). The original assumptions will be returned to the resulting subgoals using *check_asm_tac*.

Tactic

$$\frac{\{\ \Gamma\ \}\ t}{\begin{array}{c} current\_ad\_pr\_tac()thms(\{\},\ t) \\ THEN\ MAP\_EVERY\ check\_asm\_tac\ \Gamma \end{array}}$$

prove_tac
thms

See Also *PC_T1* to defer accessing the proof context until application to the goal; and, *asm_prove_tac* for the form that does react to the presence of assumptions.

Errors

51021 The current proof context was created in theory ?0 at a point now either not in scope, deleted or modified

and as the proof context setting.

It is possible that if *prove_tac* does not prove all its subgoals, then there may be an identification of newly introduced variables with free variables in the assumptions that were temporarily put to one side. This will lead to failures in the execution of the proof parts of the tactics that constitute the current proof context's automatic prover. Such a failure may not give particularly helpful messages concerning the cause of the failure. The problem is avoided by using *asm_prove_tac*, or by a call to *rename_tac* to change the offending variable names.

SML
```
val prove_thm : (string * TERM * TACTIC) -> THM;
```

Description *prove_thm* (*key*, *gl*, *tac*) applies the tactic *tac* to the goal ([], *tm*), and, if the tactic succeeds in proving the goal saves the theorem under the key given, and returns the resulting theorem.

prove_thm performs α-conversion as necessary to ensure that the theorem returned has the same form as the specified goal. In circumstances where these adjustments are known not to be necessary, *simple_tac_proof* may be used to avoid the overhead.

Defn
```
prove_thm (key, tm, tac) = save_thm(key, tac_proof(([],tm).tac));
```

Uses The subgoal package is the normal interactive mechanism for developing proofs using tactics. *prove_thm* is typically used in tactic programming and other proof procedures, in cases where it is necessary to ensure that the correct goal is proved and saved.

Errors

9501	*the tactic returned unsolved subgoals: ?0*
9502	*evaluation of the tactic failed: ?0*
9503	*the proof returned by the tactic failed: ?0*
9504	*the proof returned by the tactic proved ?0 which could not be converted into the desired goal.*
9507	*the conclusion ?0 is not of type ⌜:BOOL⌝*

See Also *simple_tac_proof*, *prove_thm*.

SML
```
val prove_∃_tac : TACTIC;
```

Description This tactic is an automatic proof procedure for existential proofs, appropriate to the current proof context.

At the point of applying this tactic to a goal it will access the current setting of proof context field *prove_∃*, apply it to the goal, with its assumptions temporarily removed, using *conv_tac*. The original assumptions will be returned to the resulting subgoals using *check_asm_tac*.

Tactic

$$\frac{\{\ \Gamma\ \}\ t}{\begin{array}{l} conv\_tac(current\_ad\_cs\_\exists\_conv())(\{\},\ t) \\ THEN\ MAP\_EVERY\ check\_asm\_tac\ \Gamma \end{array}} \qquad \begin{array}{l} prove\_\exists\_tac \\ thms \end{array}$$

See Also *asm_prove_∃_tac* that does react to any assumptions that are present.

Errors

51021	*The current proof context was created in theory ?0 at a point now either not in scope, deleted or modified*

and as the proof context setting.

SML
```
val rename_tac : (TERM * string)list -> TACTIC;
```

Description *rename_tac* renames variables (bound or free) in a goal. It is typically used when a goal contains several variables with the same name or to introduce names which are better mnemonics. For the latter purpose, the argument controls the algorithm used to make variants of the names.

The renaming affects both the conclusions and the assumptions of the goal. Variables are renamed to ensure that the new goal satisfies the following conditions:

- No two free variables with different types have the same name.
- No bound variable has the same name as a free variable or a variable which is bound in an outer scope.
- No variable shall have the same name as any constant in scope.

Before a variable is checked, it is looked up in the *renaming* association list, and if present it is treated as if the name were the corresponding string. The function *variant*, q.v., is used to rename variables.

The function may be partially evaluated with only the renamings argument.

Note that applying the tactic in the subgoal package will give rise to the message "The subgoal <label> is α-convertible to its goal".

For example,

Tactic
```
{ k = 1 }
(∀ i:ℕ×ℕ• ∃i:ℕ•i = 0)
∧ (∀ j:ℕ×ℕ
 • (∃k:ℕ•k = Fst j)
   ∧ ∀j:ℕ•j = k)
=================================
{ apple = 1 }
(∀ i• ∃ i′• i′ = 0)
∧ (∀ apple
   • (∃ carrot′• carrot′ = Fst apple)
     ∧ ∀ banana• banana = carrot)
```

rename_tac
[(⌜j:ℕ×ℕ⌝,"apple"),
(⌜j:ℕ⌝, "banana"),
(⌜k:ℕ⌝, "carrot")]

Uses In clarifying goals where the variable names clash or are unparseable or are inconvenient.

Errors
```
3007   ?0 is not a term variable
```

SML
```
val REPEAT_N_T : int -> TACTIC -> TACTIC;
val REPEAT_N : int -> TACTIC -> TACTIC;
```

Description *REPEAT_N_T n* is a tactical which repeatedly applies its tactic argument *n* times. Unlike *REPEAT* it fails if the tactic fails. If *n* is not greater than 0 then *REPEAT_N_T n tac* is a tactic which has no effect.

REPEAT_N is an alias for *REPEAT_N_T*.

Errors As for the tactic being repeated.

SML
```
val REPEAT_TTCL : THM_TACTICAL -> THM_TACTICAL;
```

Description *REPEAT_TTCL ttcl* is a theorem tactical which applies *ttcl* repeatedly until it fails.

Uses For use in programming theorem tacticals. As for the argument theorem tactic.

SML
```
val REPEAT_T : TACTIC -> TACTIC;
val REPEAT : TACTIC -> TACTIC;
```

Description *REPEAT_T* is a tactical which repeatedly applies its tactic argument until it fails. This may cause an infinite loop of evaluation, or even no change, if the tactic fails on the first application. *REPEAT* is an alias for *REPEAT_T*.

SML
```
val REPEAT_UNTIL_T1 : (GOAL -> bool) -> TACTIC -> TACTIC;
val REPEAT_UNTIL1 : (GOAL -> bool) -> TACTIC -> TACTIC;
```

Description *REPEAT_UNTIL1_T1 p tac* is a tactical which repeatedly applies its *tac* until all outstanding subgoals either satisfy the predicate *p* or cause *tac* to fail.

If the goal already satisfies *p*, then *REPEAT_UNTIL1_T1 p tac* is a tactic which has no effect.

REPEAT_UNTIL1 is an alias for *REPEAT_UNTIL1_T1*.

Example
```
REPEAT_UNTIL1 (is_or o snd) strip_tac
```

will repeatedly apply *strip_tac* until all outstanding subgoals have disjunctive conclusions or cause *strip_tac* to fail.

SML
```
val REPEAT_UNTIL_T : (TERM -> bool) -> TACTIC -> TACTIC;
val REPEAT_UNTIL : (TERM -> bool) -> TACTIC -> TACTIC;
```

Description *REPEAT_UNTIL1_T p tac* is a tactical which repeatedly applies its *tac* until all outstanding subgoals either have conclusions which satisfy the predicate *p* or cause *tac* to fail.

If the conclusion of the goal already satisfies *p*, then *REPEAT_UNTIL1_T1 p tac* is a tactic which has no effect.

REPEAT_UNTIL is an alias for *REPEAT_UNTIL_T*.

Example
```
REPEAT_UNTIL is_or strip_tac
```

will repeatedly apply *strip_tac* until all outstanding subgoals have disjunctive conclusions or cause *strip_tac* to fail.

SML
```
val rewrite_tac : THM list -> TACTIC;
val pure_rewrite_tac : THM list -> TACTIC;
val once_rewrite_tac : THM list -> TACTIC;
val pure_once_rewrite_tac : THM list -> TACTIC;
val asm_rewrite_tac : THM list -> TACTIC;
val pure_asm_rewrite_tac : THM list -> TACTIC;
val once_asm_rewrite_tac : THM list -> TACTIC;
val pure_once_asm_rewrite_tac : THM list -> TACTIC;
```

Description These are the rewriting tactics. They use the canonicalisation rule held by the current proof context (see, e.g., *push_pc*) to preprocess the theorem list. The context is accessed at the point when the rules are given a list of theorems.

If a tactic is "pure" then there is no default rewriting, otherwise the default rewriting conversion net held by the current proof context will be used in addition to user supplied material.

If a tactic is "once" then rewriting will proceed from the root of the of the conclusion of the theorem to be rewritten, towards the leaves, and will not descend through any rewritten subterm, using *ONCE_MAP_WARN_C*. If not, rewriting will continue, moving from the root to the leaves, repeating if any rewriting is successful, until there is no rewriting redex anywhere within the rewritten conclusion, using *REWRITE_MAP_C*. This may cause non-terminating looping.

If a tactic is "asm" then the theorems rewritten with will include the canonicalised *asm_rule*d assumptions of the goal.

Errors
```
26001 no rewriting occurred
```

Also as error 26003 and warning 26002 of *REWRITE_MAP_C* (q.v.).

SML
```
val rewrite_thm_tac : THM -> TACTIC;
val pure_rewrite_thm_tac : THM -> TACTIC;
val once_rewrite_thm_tac : THM -> TACTIC;
val pure_once_rewrite_thm_tac : THM -> TACTIC;
val asm_rewrite_thm_tac : THM -> TACTIC;
val pure_asm_rewrite_thm_tac : THM -> TACTIC;
val once_asm_rewrite_thm_tac : THM -> TACTIC;
val pure_once_asm_rewrite_thm_tac : THM -> TACTIC;
```

Description These are rewriting tactics parameterised to take only one theorem. This parameterisation is convenient to use with the many tactic generating functions, such as *LEMMA_T*, which take a theorem tactic as an argument.

See, e.g. *rewrite_tac* for the details of the differences between these tactics.

Errors
```
26001 no rewriting occurred
```

Errors will be reported as if they are from the corresponding *_tac*: e.g. from *rewrite_tac* rather than *rewrite_thm_tac*. This allows a simple implementation, and for there to be no functionality change even in errors between using singleton lists with the originals, and these functions. The following warning indicates the result of, perhaps only some, of the rewriting was discarded.

Errors
```
26002 rewriting gave ill-formed results on some subterms
```

SML
```
val ROTATE_T : int -> TACTIC -> TACTIC;
```

Description *ROTATE_T i tac* is a tactic which first applies *tac* and, if this does not achieve the goal, rotates the resulting subgoals by *i* places. *i* is taken modulo the number of subgoals produced by *tac*.

Thus if the result of *tac* is:

Tactic

$$\frac{\{\ \Gamma\ \}\ t}{\{\ \Gamma 1\ \}\ t1;\ \dots\ \{\ \Gamma k\ \}\ tk} \qquad tac$$

then the result of *ROTATE_T i t* will be:

Tactic

$$\frac{\{\ \Gamma\ \}\ t}{\begin{array}{c}\{\ \Gamma(i{+}1)\ \}\ t(i{+}1);\ \dots,\ \{\ \Gamma k\ \}\ tk;\\ \{\ \Gamma 1\ \}\ t1;\ \dots\ \{\ \Gamma i\ \}\ ti\end{array}} \qquad ROTATE\_T\ i\ tac$$

Uses For use in tactic programming to handle tactics which return their subgoals in an inconvenient order for the task at hand.

Errors As for *tac*.

SML
```
val simple_tac_proof : (GOAL * TACTIC) -> THM;
```

Description *simple_tac_proof*(*gl*, *tac*) applies the tactic *tac* to the goal *gl*, and, if the tactic returns no unsolved subgoals returns the theorem proved by the tactic.

Infelicities in the coding of the tactic may cause the theorem returned to be rather different from the specified goal (in general, a "successful" application of a correctly coded tactic will return a theorem which may require addition of assumptions and α-conversion to give the desired goal). *tac_proof* should be used rather than *simple_tac_proof* if it is important that the theorem should achieve the goal precisely.

Uses In programming tactics or other proof procedures where speed is important and the extra care taken by *tac_proof* is not required.

Errors

9501	*the tactic returned unsolved subgoals: ?0*
9502	*evaluation of the tactic failed: ?0*
9503	*the proof returned by the tactic failed: ?0*

See Also *tac_proof*

SML
```
val simple_taut_tac : TACTIC;
```

Description A tautology prover. If the conclusion of the goal is a tautology then *taut_tac* will prove the goal. A tautology is taken to be any substitution instance of a term which is formed from boolean variables, the constants T and F and the following connectives:

$\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\neg$, *if ... then ... else*

and which is true for any assignment of truth values to the variables.

Tactic

$$\frac{\{\ \Gamma\ \}\ t}{} \qquad simple\_taut\_tac$$

See Also *strip_tac*

Errors

28121 Conclusion of the goal is not a tautology

SML
```
val simple_¬_in_conv : CONV;
```

Description This is a conversion which moves negations inside other predicate calculus connectives using whichever of the following rules applies:

$$\begin{array}{lcl}
\neg\neg t & = & t \\
\neg(t1 \wedge t2) & = & \neg t1 \vee \neg t2 \\
\neg(t1 \vee t2) & = & \neg t1 \wedge \neg t2 \\
\neg(t1 \Rightarrow t2) & = & t1 \wedge \neg t2 \\
\neg(t1 \Leftrightarrow t2) & = & (t1 \wedge \neg t2) \vee (t2 \wedge \neg t1) \\
\neg(if\ a\ then\ t1\ else\ t2) & & \\
 & = & (if\ a\ then\ \neg t1\ else\ \neg t2) \\
\neg\forall x \bullet t & = & \exists x \bullet \neg t \\
\neg\exists x \bullet t & = & \forall x \bullet \neg t \\
\neg\exists_1 x \bullet t = & \forall x \bullet \neg(t \wedge \forall x' \bullet t[x'] \Rightarrow x' = x) & \\
\neg T & = & F \\
\neg F & = & T
\end{array}$$

It does not handle paired quantifiers.

Uses Tactic and conversion programming. The more general *¬_in_conv* is just as efficient as *simple_¬_in_conv* in cases where both succeed.

See Also *strip_tac*

Errors

28131 No applicable rules for the term ?0

SML
```
val simple_¬_in_tac : TACTIC;
```

Description This is a tactic which moves negations inside other predicate calculus connectives using the following rules:

$\neg\neg t$	$\rightarrow$	t
$\neg(t1 \wedge t2)$	$\rightarrow$	$\neg t1 \vee \neg t2$
$\neg(t1 \vee t2)$	$\rightarrow$	$\neg t1 \wedge \neg t2$
$\neg(t1 \Rightarrow t2)$	$\rightarrow$	$t1 \wedge \neg t2$
$\neg(t1 \Leftrightarrow t2)$	$\rightarrow$	$(t1 \wedge \neg t2) \vee (t2 \wedge \neg t1)$
$\neg\forall x\bullet t$	$\rightarrow$	$\exists x\bullet\neg t$
$\neg\exists x\bullet t$	$\rightarrow$	$\forall x\bullet\neg t$
$\neg\exists_1 x\bullet t\rightarrow$	$\forall x\bullet\neg(t \wedge \forall x'\bullet t[x'] \Rightarrow x' = x)$	
$\neg T$	$\rightarrow$	F
$\neg F$	$\rightarrow$	*goal solved*

It fails with paired quantifiers.

Uses The more general *¬_in_tac* is just as efficient as *simple_¬_in_tac* in cases where both succeed.

See Also *strip_tac*, *contr_tac*, *¬_T*, *¬_in_tac*

Errors
```
28025 No applicable rule for this goal
```

SML
```
val SIMPLE_¬_IN_THEN : THM_TACTICAL;
```

Description This is a theorem tactical which applies a given theorem tactic to the result of transforming a theorem by moving a top level negation inside other predicate calculus connectives using the following rules:

$\neg\neg t$	$\rightarrow$	t
$\neg(t1 \wedge t2)$	$\rightarrow$	$\neg t1 \vee \neg t2$
$\neg(t1 \vee t2)$	$\rightarrow$	$\neg t1 \wedge \neg t2$
$\neg(t1 \Rightarrow t2)$	$\rightarrow$	$t1 \wedge \neg t2$
$\neg(t1 \Leftrightarrow t2)$	$\rightarrow$	$(t1 \wedge \neg t2) \vee (t2 \wedge \neg t1)$
$\neg\forall x\bullet t$	$\rightarrow$	$\exists x\bullet\neg t$
$\neg\exists x\bullet t$	$\rightarrow$	$\forall x\bullet\neg t$
$\neg\exists_1 x\bullet t\rightarrow$	$\forall x\bullet\neg(t \wedge \forall x'\bullet t[x'] \Rightarrow x' = x)$	
$\neg T$	$\rightarrow$	F
$\neg F$	$\rightarrow$	T

The function may be partially evaluated with only its theorem tactic and theorem arguments. It fails with paired quantifiers.

Uses The more general *¬_IN_THEN* is just as efficient as *SIMPLE_¬_IN_THEN* in cases where both succeed.

See Also *strip_tac*, *STRIP_THM_THEN*

Errors
```
28026 No applicable rule for this theorem
```

SML

```
val simple_∀_tac : TACTIC;
```

Description Reduce a universally quantified goal. It fails with paired quantifiers.

Tactic

$$\frac{\{\ \Gamma\ \}\ \forall\ x\ \bullet\ t[x]}{\{\ \Gamma\ \}\ t[x']} \qquad \textit{simple\_}\forall\textit{\_tac}$$

where x' is a variant name of x, different from any variable in Γ or t.

Uses Tactic programming. The more general *∀_tac* is just as efficient as *simple_∀_tac* in cases where both succeed.

See Also *∀_tac*

Errors

28081 Goal is not of the form: { Γ } ∀ x • $t[x]$

SML

```
val simple_∃_tac : TERM -> TACTIC ;
```

Description Provide a witness for an existential subgoal. It fails with paired quantifiers.

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

Tactic

$$\frac{\{\ \Gamma\ \}\ \exists\ x\ \bullet\ t2[x]}{\{\ \Gamma\ \}\ t2[t1]} \qquad \begin{array}{l}\textit{simple\_}\exists\textit{\_tac}\\ \ulcorner t1 \urcorner\end{array}$$

where *t1* must have the same type as x.

Uses Tactic programming. The more general *∃_tac* is just as efficient as *simple_∃_tac* in cases where both succeed.

Errors

28091 Goal is not of the form: { Γ } ∃ x • $t2[x]$
28092 Term ?0 has the wrong type

SML

val **SIMPLE_∃_THEN** : $(THM \text{ -> } TACTIC) \text{ -> } (THM \text{ -> } TACTIC)$;

Description A theorem tactical which applies a given theorem tactic to the result of eliminating the outermost quantifier from a theorem of the form $\Gamma \vdash \exists x \bullet t$. It fails with paired quantifiers.

$SIMPLE\_\exists\_THEN\ thmtac\ (\Gamma \vdash \exists x \bullet t) = thmtac\ (\Gamma \vdash t[x'/x])$

where ⌜x'⌝ is a variant of ⌜x⌝ which does not appear in Γ or in the assumption or conclusion of the goal. The function is partially evaluated with only the theorem tactic and theorem arguments.

Uses Tactic programming. Note that the more general *∃_THEN* is just as efficient as *SIMPLE_∃_THEN* in cases where both succeed.

Error 28094 normally arises when ⌜x'⌝ is also introduced by the proof of *ttac*, and occurs during the application of the proof of *SIMPLE_∃_THEN*. The bound variable ⌜x⌝ should be renamed to something that doesn't cause this identification of distinct variables, by using *rename_tac*(q.v.).

See Also *∃_THEN*

Errors

28093 ?0 is not of the form: '$\Gamma \vdash \exists\ x \bullet t$'
28094 Error in proof of SIMPLE_∃_THEN.
Usually indicates chosen skolem variable ?0 also
introduced by proof of supplied theorem tactic,
which gave '?1', and the two became identified:
use rename_tac to rename original bound variable ?2

SML

val **simple_∃₁_conv** : $CONV$;

Description This is a conversion which turns a unique existential quantifier into an equivalent existential quantifier

Conversion

$$\frac{}{\vdash (\exists_1 x \bullet t[x]) \Leftrightarrow (\exists x \bullet t[x] \land \forall x' \bullet t[x'] \Rightarrow x' = x)} \quad \begin{array}{l} simple\_\exists_1\_conv \\ ⌜\exists_1 x \bullet t⌝ \end{array}$$

Uses Tactic and conversion programming. The more general *∃_1_conv* is just as efficient as *simple_∃_1_conv* in cases where both succeed.

See Also *strip_tac*

Errors

4019 ?0 is not of form: ⌜$\exists_1\ v \bullet\ t$⌝

SML
```
val simple_∃₁_tac : TERM -> TACTIC;
```

Description Simplify a unique existentially quantified goal with a particular witness. It fails with paired quantifiers.

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

Tactic

$$\frac{\{\ \Gamma\ \}\ simple\_\exists_1\ x \bullet P[x]}{\{\ \Gamma\ \}\ P[t];\quad \{\ \Gamma\ \}\ \forall\ x' \bullet P[x'] \Rightarrow x' = t} \qquad simple\_\exists_1\_tac1\ \ulcorner t \urcorner$$

where x' is a variant of x which does not occur free in t.

Uses Tactic programming. The more general *∃_1_tac* is just as efficient as *simple_∃_1_tac* in cases where both succeed.

Errors

28101 Goal is not of the form: { Γ } ∃₁ x • P[x]
28092 Term ?0 has the wrong type

SML
```
val SIMPLE_∃₁_THEN : (THM -> TACTIC) -> (THM -> TACTIC);
```

Description A theorem tactical which applies a given theorem tactic to the result of eliminating the outermost quantifier from a theorem of the form $\Gamma \vdash \exists\_1x \bullet t$. It fails with paired quantifiers.

$$SIMPLE\_\exists_1\_THEN\ thmtac\ (\Gamma \vdash \exists_1 x \bullet t) =$$
$$\quad thmtac\ (\Gamma \vdash t[x'/x] \wedge \forall x'' \bullet P[x''] \Rightarrow x'' = x)$$

where ⌜x'⌝ and ⌜x''⌝ are distinct variants of ⌜x⌝ which do not appear free in Γ or in the assumptions or conclusion of the goal.

Uses Tactic programming. The more general *∃₁_THEN* is just as efficient as *SIMPLE-_∃₁_THEN* in cases where both succeed.

Errors

28102 ?0 is not of the form: ‘Γ ⊢ ∃₁ x • t‘

SML
```
val SOLVED_T : TACTIC -> TACTIC;
```

Description *SOLVED_T tac* is a tactic which applies *tac* to the goal and fails if it does not solve the goal. I.e. it fails unless the tactic returns an empty list of subgoals.

SOLVED_T does not check that the proof delivered by the tactic is valid. *tac_proof* may be used to achieve this type of effect.

Uses Tactic programming, for when a tactic that fails to prove a goal is likely to leave an untidy goal state.

See Also *tac_proof*

Errors

9602 the tactic did not solve the goal

SML

val **spec_asm_tac** : $TERM$ −> $TERM$ −> $TACTIC$;
val **list_spec_asm_tac** : $TERM$ −> $TERM$ $list$ −> $TACTIC$;
val **spec_nth_asm_tac** : int −> $TERM$ −> $TACTIC$;
val **list_spec_nth_asm_tac** : int −> $TERM$ $list$ −> $TACTIC$;

Description These are four methods of specialising assumptions, differing by single or lists of values to specialise to, and in the method of selection of the assumption. All of them leave the old assumption in place, and place the instantiated assumption onto the assumption list using *strip_asm_tac*. If the desired behaviour differs from any of those supplied then use *GET_ASM_T* and its cousins to create the desired functionality.

Tactic

$$\frac{\{\ \Gamma,\ \ulcorner\forall\ vs[x1',...]\ \bullet\ f\ [x1',...]\urcorner\ \}\ t}{\{strip\ \ulcorner f\ [t1,...]\urcorner,\ \Gamma,\ \ulcorner\forall\ vs[x1,...]\ \bullet\ f\ [x1,...]\urcorner\}\ t1}\qquad \begin{array}{l} spec\_asm\_tac \\ \ulcorner\forall\ vs[x1,...]\ \bullet\ f\ [x1,...]\urcorner \\ \ulcorner tm\urcorner \end{array}$$

The following all handle paired abstractions in a similar manner.

Tactic

$$\frac{\{\ \Gamma,\ \ulcorner\forall\ x1\ ...\ \bullet\ f\ [x1,...]\urcorner\ \}\ t}{\{strip\ \ulcorner f\ [t1,...]\urcorner,\ \Gamma,\ \ulcorner\forall\ x1\ x2\ ...\ \bullet\ f\ [x1,...]\urcorner,\ \}\ t}\qquad \begin{array}{l} list\_spec\_asm\_tac \\ \ulcorner\forall\ x1\ ...\ \bullet\ f\ [x1,...]\urcorner \\ [\ulcorner t1\urcorner,\ulcorner t2\urcorner,...] \end{array}$$

Tactic

$$\frac{\{\ \Gamma 1...n{-}1,\ \ulcorner\forall\ x'\ \bullet\ f\ [x']\urcorner,\ \Gamma\}\ t1}{\{strip\ \ulcorner f\ [t2]\urcorner,\ \ \Gamma 1...n{-}1,\ \ulcorner\forall\ x'\ \bullet\ f\ [x']\urcorner,\ \Gamma\}\ t1}\qquad \begin{array}{l} spec\_nth\_asm\_tac \\ n \\ \ulcorner t2\urcorner \end{array}$$

Tactic

$$\frac{\{\ \Gamma 1...n{-}1,\ \ulcorner\forall\ x1\ ...\bullet\ f\ [x1,...]\urcorner,\ \Gamma\}\ t}{\{strip\ \ulcorner f\ [t1,...]\urcorner,\ \Gamma 1...n{-}1,\ \ulcorner\forall\ x1\ ...\bullet\ f\ [x1,...]\urcorner,\ \Gamma\}\ t}\qquad \begin{array}{l} list\_spec\_nth\_asm\_tac \\ n \\ [\ulcorner t1\urcorner,...] \end{array}$$

Definitions

fun spec_asm_tac asm instance =
    GET_ASM_T asm (strip_asm_tac o ∀_elim instance);
fun list_spec_asm_tac asm instances =
    GET_ASM_T asm (strip_asm_tac o list_∀_elim instances);
fun spec_nth_asm_tac n instance =
    GET_NTH_ASM_T n (strip_asm_tac o ∀_elim instance);
fun list_spec_nth_asm_tac n instances =
    GET_NTH_ASM_T n (strip_asm_tac o list_∀_elim instances);

Errors As the constituents of the implementing functions.

SML

```
val SPEC_ASM_T : TERM -> TERM -> (THM -> TACTIC) -> TACTIC;
val LIST_SPEC_ASM_T : TERM -> TERM list -> (THM -> TACTIC)
        -> TACTIC;
val SPEC_NTH_ASM_T : int -> TERM -> (THM -> TACTIC) -> TACTIC;
val LIST_SPEC_NTH_ASM_T : int -> TERM list -> (THM -> TACTIC)
        -> TACTIC;
```

Description These are four methods of specialising assumptions, differing by single or lists of values to specialise to, and in the method of selection of the assumption. All of them leave the old assumption in place, and place the instantiated assumption onto the assumption list using their theorem tactic. If the desired behaviour differs from any of those supplied then use *GET_ASM_T* and its cousins to create the desired functionality.

Tactic

$$\frac{\{\ \Gamma,\ \ulcorner\forall\ vs[x1',...]\ \bullet\ f\ [x1',...]\urcorner\ \}\ t}{\overline{\begin{array}{c} thm\_tac\ (asm\_rule\ \ulcorner f\ [t1,...]\urcorner) \\ \{\Gamma,\ulcorner\forall\ vs[x1,...]\ \bullet\ f\ [x1,...]\urcorner\}\ t1\end{array}}}$$

SPEC_ASM_T
$\ulcorner\forall\ vs[x1,...]\ \bullet\ f\ [x1,...]\urcorner$
$\ulcorner tm\urcorner$
thm_tac

The following all handle paired abstractions in a similar manner.

Tactic

$$\frac{\{\ \Gamma,\ \ulcorner\forall\ x1\ \ x2\ \ ...\ \bullet\ f\ [x1,x2,...]\urcorner\ \}\ t}{\overline{\begin{array}{c} thm\_tac\ (asm\_rule\ \ulcorner f\ [t1,t2,...]\urcorner) \\ \{\Gamma,\ulcorner\forall\ x1\ \ x2\ \ ...\ \bullet\ f\ [x1,x2,...]\urcorner\}\ t\end{array}}}$$

LIST_SPEC_ASM_T
$\ulcorner\forall\ x1\ \ x2\ \ ...\ \bullet\ f\ [x1,x2,...]\urcorner$
$[\ulcorner t1\urcorner,\ulcorner t2\urcorner,...]$
thm_tac

Tactic

$$\frac{\{\ \Gamma 1...n{-}1,\ \ulcorner\forall\ x'\ \bullet\ f\ [x']\urcorner,\ \Gamma\}\ t1}{\overline{\begin{array}{c} thm\_tac\ (asm\_rule\ \ulcorner f\ [t2]\urcorner) \\ \{\Gamma 1...n{-}1,\ulcorner\forall\ x'\ \bullet\ f\ [x']\urcorner,\Gamma\}\ t1\end{array}}}$$

SPEC_NTH_ASM_T
n
$\ulcorner t2\urcorner$
thm_tac

Tactic

$$\frac{\{\ \Gamma 1...n{-}1,\ \ulcorner\forall\ x1\ \ ...\bullet\ f\ [x1,...]\urcorner,\ \Gamma\}\ t}{\overline{\begin{array}{c} thm\_tac\ (asm\_rule\ \ulcorner f\ [t1,...]\urcorner) \\ \{\Gamma 1...n{-}1,\ulcorner\forall\ x1\ \ ...\bullet\ f\ [x1,...]\urcorner,\ \Gamma\} \\ t\end{array}}}$$

LIST_SPEC_NTH_ASM_T
n
$[\ulcorner t1\urcorner,...]$
thm_tac

Definitions

```
fun SPEC_ASM_T asm instance thmtac =
        GET_ASM_T asm (thmtac o ∀_elim instance);
fun LIST_SPEC_ASM_T asm instances thmtac =
        GET_ASM_T asm (thmtac o list_∀_elim instances);
fun SPEC_NTH_ASM_T n instance thmtac =
        GET_NTH_ASM_T n (thmtac o ∀_elim instance);
fun LIST_SPEC_NTH_ASM_T n instances thmtac =
        GET_NTH_ASM_T n (thmtac o list_∀_elim instances);
```

Errors As the constituents of the implementing functions.

SML

```
val step_strip_tac : TACTIC;
val step_strip_asm_tac : THM -> TACTIC;
```

Description These functions provide methods of single-stepping through the application of *strip_tac* and *strip_asm_tac* (q.v.).

When stripping the antecedent of an implication ,or a theorem, into the assumption list *strip_tac* and *strip_asm_tac* respectively do all their stripping in one application of the tactic. This is not appropriate behaviour when:

1. Explaining the detailed behaviour of these functions by example applications.
2. Attempting to "debug" a failed or inappropriate stripping.
3. When a partial strip into the assumption list is desired.

The two functions provided give a single-step stripping of antecedents and theorems. They represent sets of objects that are partially stripped into the assumption list by making the conclusion of the resulting goal an implication with the antecedent being the conjunction of the partially stripped objects and the consequent being the unstripped part of the goal. Repeated use of the provided functions closely corresponds to the processing order and effect of *strip_tac* and *strip_asm_tac*. Under certain unusual circumstances the match may not be exact.

Example

$\vdash ?\ ((a \lor b) \land c) \Rightarrow ((a \land c) \lor (b \land c))$
Single steps to:
$\vdash ?\ (a \land c) \Rightarrow ((a \land c) \lor (b \land c))$
and $\vdash ?\ (b \land c) \Rightarrow ((a \land c) \lor (b \land c))$
Each single step to:
$a \vdash ?\ c \Rightarrow ((a \land c) \lor (b \land c))$
and $b \vdash ?\ c \Rightarrow ((a \land c) \lor (b \land c))$
Each single step to:
$a,\ c \vdash ? \Rightarrow ((a \land c) \lor (b \land c))$
and $b,\ c \vdash ? \Rightarrow ((a \land c) \lor (b \land c))$

These five steps (two on each branch) map onto one call of *strip_tac*.

Errors

28003 There is no stripping technique for ?0 in the current proof context

SML

```
val strip_asm_tac : THM -> TACTIC;
```

Description *strip_asm_tac* is a general purpose tactic for splitting a theorem up into useful pieces using a range of simplification techniques, including a parameterised part, before using it to increase the stock of assumptions.

First, before attempting to use the transformations below, *strip_asm_tac* uses the current proof context's theorem stripping conversion to attempt to rewrite the outermost connective in the theorem.

Then the following simplification techniques will be tried. Using *sat* as an abbreviation for *strip_asm_tac*:

$sat\ (\vdash a \land b)$	$\rightarrow$	$sat\ (\vdash a)\ THEN\ sat\ (\vdash b)$
$sat\ (\exists x \bullet a)$	$\rightarrow$	$sat\ (a[x'/x] \vdash a[x'/x])$
$sat\ (\vdash a \lor b)(\{\Gamma\}\ t)$	$\rightarrow$	$sat\ (a \vdash a)\ (\{\Gamma\}\ t)\ ;\ sat\ (b \vdash b)\ (\{\Gamma\}\ t)$

I.e. *strip_asm_tac* does a case split resulting in two subgoals when it processes a disjunction.

After all of the available simplification techniques have been attempted *strip_asm_tac* then proceeds as *check_asm_tac* (q.v.) to use the simplified theorem either to prove the goal or to generate additional assumptions.

See Also *STRIP_THM_THEN*, used to implement this function. *check_asm_tac*, *strip_tac*, *strip_asm_conv*.

SML

```
val strip_concl_conv : CONV;
val strip_asm_conv : CONV;
```

Description *strip_concl_conv tm*; applies the conclusion stripping conversion from the current proof context, to rewrite the outermost connective in the term *tm*.

strip_asm_conv tm; applies the assumption stripping conversion from the current proof context, to rewrite the outermost connective in the term *tm*.

Errors

```
28003 There is no stripping technique for ?0 in the current proof context
```

SML
```
val strip_concl_tac : TACTIC;
val strip_tac : TACTIC;
```

Description *strip_concl_tac*, more usually known by its alias, *strip_tac*, is a general purpose tactic for simplifying away the outermost connective of a goal. It first tries to apply the conclusion stripping conversion from the current proof context, to rewrite the outermost connective in the goal. If that conversion fails, tries to simplify the goal by applying an applicable member of the following collection of tactics (only one could possibly apply):

```
simple_∀_tac,          ∧_tac,
⇒_T strip_asm_tac,     t_tac
```

Failing either being successful, it tries *concl_in_asms_tac* to prove the goal, and failing that, returns the error message below.

Note how new assumptions generated by the tactic are processed using *strip_asm_tac*, which uses the current proof context's theorem stripping conversion. *strip_tac* may produce several new subgoals, or may prove the goal.

REPEAT strip_tac in the proof context "basic_hol" (amongst others) will prove all tautologies automatically. It will, however, not succeed in proving some substitution instances of tautologies involving positive and negative instances of a quantified subterm.

Uses This is the usual way of simplifying a goal involving predicate calculus connectives, and other functions "understood" by the current prof context.

See Also *STRIP_T* and *STRIP_THM_THEN* which are used to implement this function. *taut_tac* for an alternative simplifier. *swap_∨_tac* to rearrange the conclusion for tailored stripping. Also *strip_concl_conv*, *strip_asm_conv*.

Errors
```
28003 There is no stripping technique for ?0 in the current proof context
```

SML
```
val STRIP_CONCL_T : (THM -> TACTIC) -> TACTIC;
val STRIP_T : (THM -> TACTIC) -> TACTIC;
```

Description *STRIP_CONCL_T ttac* is a general purpose way of stripping goals and passing any new assumptions generated by the stripping to a tactic generating function, *ttac*. *STRIP-_CONCL_T* attempts to apply the conversion held for it in the current proof context to rewrite the goal. The conversion is extracted from the current proof context by *current_ad_sc_conv*. If that fails it attempts to apply one of the following list of tactics (in order):

```
simple_∀_tac, ∧_tac, ⇒_T ttac,     t_tac
```

If none of the above apply it tries *concl_in_asms_tac*, and failing that, return the error message below.

The conversion in the current proof context held by *current_ad_sc_conv* (q.v.) is derived by applying *eqn_cxt_conv* to an equational context in the proof context, extracted by *get_sc_eqn_cxt*.

STRIP_T is an alias for *STRIP_CONCL_T*.

Uses Tactic programming.

See Also *strip_asm_tac*, *strip_tac*, *strip_concl_conv*.

Errors
```
28003 There is no stripping technique for ?0 in the current proof context
```

SML
```
val STRIP_THM_THEN : THM_TACTICAL;
```

Description *STRIP_THM_THEN* provides a general purpose way of stripping theorems into primitive constituents before using them in a tactic proof. *STRIP_THM_THEN* attempts to apply the conversion held for the function in the current proof context, which is extracted by *current_ad_st_conv*. to rewrite the theorem. If that fails it attempts to apply a theorem tactical from the following list (in order):

```
∧_THEN,          ∨_THEN,          SIMPLE_∃_THEN
```

The conversion in the current proof context got by *current_ad_st_conv* (q.v.) is derived by applying *eqn_cxt_conv* to an equational context in the proof context extracted by *get_st_eqn_cxt*.

The function is partially evaluated with only the theorem tactic and theorem arguments.

Uses Tactic programming.

See Also *strip_asm_tac*, *strip_tac*.

Errors
```
28003 There is no stripping technique for ?0 in the current proof context
```

SML
```
val swap_asm_concl_tac : TERM -> TACTIC;
val swap_nth_asm_concl_tac : int -> TACTIC;
```

Description Strip the negation of current goal into the assumption list and make an assumption, suitably negated, into the current goal. If the simplifications it does are ignored, *swap_asm_concl_tac asm*is equivalent to

Example
```
contr_tac THEN asm_ante_tac asm
```

and *swap_nth_asm_concl_tac n*is equivalent to

Example
```
contr_tac THEN DROP_NTH_ASM_T n ante_tac
```

Tactic

$$\frac{\{\ \Gamma,\ \ulcorner t1 \urcorner\ \}\ t2}{\{strip\ \ulcorner\neg\ t2\urcorner,\ \Gamma\}\ \neg\ t1} \qquad swap\_asm\_concl\_tac\ \ulcorner t1 \urcorner$$

Tactic

$$\frac{\{\ \ulcorner t1\urcorner,\ ...,\ \ulcorner tm\urcorner,...,\ \ulcorner tn\urcorner\}\ t}{\{strip\ \ulcorner\neg\ t\urcorner,\ \ulcorner t1\urcorner,...,\ \ulcorner tn\urcorner\}\ \neg\ tm} \qquad swap\_nth\_asm\_concl\_tac\ m$$

If the assumption is a negated term then the double negation will be eliminated.

See Also Other *swap* and *SWAP* functions.

Errors
```
9303  the index ?0 is out of range
28052 Term ?0 is not in the assumptions
```

SML

val **SWAP_ASM_CONCL_T** : *TERM* −> (*THM* −> *TACTIC*) −> *TACTIC*;
val **SWAP_NTH_ASM_CONCL_T** : *int* −> (*THM* −> *TACTIC*) −> *TACTIC*;

Description Process the negation of current goal with the supplied theorem tactic and make an assumption, suitably negated, into the current goal. If the simplifications it does are ignored, *SWAP_ASM_CONCL_T asm ttac*is equivalent to

Example

CONTR_T (fn x => asm_ante_tac asm THEN ttac x)

and *SWAP_NTH_ASM_CONCL_T n ttac*is equivalent to

Example

CONTR_T (fn x => (DROP_NTH_ASM_T n ante_tac) THEN ttac x)

Tactic

$$\frac{\{\ \Gamma, \ulcorner t1 \urcorner\ \}\ t2}{ttac(asm\_rule\ \ulcorner \neg t2 \urcorner\ )(\{\Gamma\}\ \neg\ t1)} \qquad SWAP\_ASM\_CONCL\_T\ \ulcorner t1 \urcorner\ ttac$$

Tactic

$$\frac{\{\ \ulcorner t1 \urcorner, ..., \ulcorner tm \urcorner, ..., \ulcorner tn \urcorner\}\ t}{ttac(asm\_rule\ \ulcorner \neg t \urcorner\ )(\{\ulcorner t1 \urcorner, ..., \ulcorner tn \urcorner\}\ \neg\ tm)} \qquad SWAP\_NTH\_ASM\_CONCL\_T\ m\ ttac$$

If the assumption is a negated term then the double negation will be eliminated.

See Also Other *swap* and *SWAP* functions.

Errors

9303 the index ?0 is out of range
28027 Conclusion of goal does not have type ⌜:*BOOL*⌝
28052 Term ?0 is not in the assumptions

SML

val **swap_∨_tac** : *TACTIC*;

Description Interchange the disjuncts of a disjunctive goal.

Tactic

$$\frac{\{\ \Gamma\ \}\ a \lor b}{\{\ \Gamma\ \}\ b \lor a} \qquad swap\_\lor\_tac$$

Uses For use in conjunction with *strip_tac* (q.v.) when the reduction of $\{\Gamma\}a \lor b$ to $\{\neg a, \Gamma\}b$ is inappropriate.

See Also *∨_left_tac*, *∨_right_tac*, *swap_∨_tac*, *strip_tac*

Errors

28041 Goal is not of the form: $\{\ \Gamma\ \}\ a \lor b$

SML
```
val tac_proof : (GOAL * TACTIC) -> THM;
```

Description *tac_proof*(*gl*, *tac*) applies the tactic *tac* to the goal *gl*, and, if the tactic succeeds in proving the goal returns the resulting theorem.

tac_proof performs α-conversion, introduces additional assumptions, and reorders assumptions as necessary to ensure that the theorem returned has the same form as the specified goal (note that this is not possible if the goal has α-equivalent assumptions). In circumstances where these adjustments are known not to be necessary, *simple_tac_proof* may be used to avoid the overhead.

Uses The subgoal package is the normal interactive mechanism for developing proofs using tactics. *tac_proof* is typically used in tactic programming and other proof procedures, in cases where it is necessary to ensure that the correct goal is proved.

Errors

9501	*the tactic returned unsolved subgoals*: ?0
9502	*evaluation of the tactic failed*: ?0
9503	*the proof returned by the tactic failed*: ?0
9504	*the proof returned by the tactic proved ?0 which could not be converted into the desired goal.*
9505	*the goal contains alpha−equivalent assumptions* (?0 *and* ?1)
9506	*the assumption ?0 is not of type* ⌜:*BOOL*⌝
9507	*the conclusion ?0 is not of type* ⌜:*BOOL*⌝

See Also *simple_tac_proof*, *prove_thm*.

SML
```
val taut_conv : CONV;
```

Description A tautology prover. A tautology is taken to be any universally quantified substitution instance of a term which is formed from boolean variables, the constants T and F and the following connectives:

$\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\neg$, *if ... then ... else*

and which is true for any assignment of truth values to the variables. If its argument is a tautologically true term, then the function will return a theorem that the term is equivalent to T.

Conversion

$$\frac{}{\vdash t \Leftrightarrow T} \quad \begin{array}{l} taut\_conv \\ \ulcorner t \urcorner \end{array}$$

See Also *taut_tac*, *taut_rule*, *simple_taut_tac*.

Errors

27037 ?0 is not tautologically true

SML
```
val taut_rule : TERM -> THM;
```

Description A tautology prover. A tautology is taken to be any universally quantified substitution instance of a term which is formed from boolean variables, the constants T and F and the following connectives:

$\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\neg$, *if ... then ... else*

and which is true for any assignment of truth values to the variables. If its argument is such a tautology then the function will return that term as a theorem.

Rule

$$\frac{}{\vdash t} \quad \begin{array}{l} taut\_rule \\ \ulcorner t \urcorner \end{array}$$

See Also *taut_tac*, *taut_conv*, *simple_taut_tac*.

Errors
```
27037 ?0 is not tautologically true
```

SML
```
val taut_tac : TACTIC;
```

Description A tautology prover. If the conclusion of the goal is a tautology then *taut_tac* will prove the goal. A tautology is taken to be any (perhaps universally quantified) substitution instance of a term which is formed from boolean variables, the constants T and F and the following connectives:

$\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\neg$, *if ... then ... else*

and which is true for any assignment of truth values to the variables.

Tactic

$$\frac{\{\ \Gamma\ \}\ t}{} \quad taut\_tac$$

See Also *strip_tac*, *taut_rule*, *taut_conv*, *simple_taut_tac*.

Errors
```
29020 Conclusion of the goal is not a universally quantified tautology
```

SML
```
val THEN_LIST_T : (TACTIC * TACTIC list) -> TACTIC;
val THEN_LIST : (TACTIC * TACTIC list) -> TACTIC;
```

Description *THEN_LIST_T* is a tactical used as an infix operator. *tac THEN_LIST_T tlist* is a tactic that applies *tac*, and then applies the first member of *tlist* to the first resulting subgoal, the second to the second, etc. If there are not the correct number of tactics in the list then an error will be raised. *THEN_LIST* is an alias for *THEN_LIST_T*.

Errors
```
9101 number of tactics must equal the number of subgoals
```

As failures of the initial tactic or the tactics in the list.

SML
```
val THEN_T1 : (TACTIC * TACTIC) -> TACTIC;
val THEN1 : (TACTIC * TACTIC) -> TACTIC;
```

Description *THEN_T1* is a tactical used as an infix operator. *tac1 THEN_T1 tac2* is the tactic that applies *tac1* and then applies *tac2* to the first of the resulting subgoals and *id_tac* to any other subgoals. If *tac1* returns no subgoals, then nor will *tac1 THEN_T1 tac2*. *THEN1* is an alias for *THEN_T1*.

It is intended for use in conjunction with induction tactics or tactics like *lemma_tac* for which the first subgoal (i.e., the base case of the induction or the lemma) often has a simple proof.

See Also *THEN*

Errors As the errors of *tac1* and *tac2*.

SML
```
val THEN_TRY_LIST_T : (TACTIC * TACTIC list) -> TACTIC;
val THEN_TRY_LIST : (TACTIC * TACTIC list) -> TACTIC;
```

Description *THEN_TRY_LIST_T* is a tactical used as an infix operator. *tac THEN_TRY_LIST_T tlist* is a tactic that applies *tac*, and then attempts to apply the first member of *tlist* to the first resulting subgoal, the second to the second, etc. If there are not the correct number of tactics in the list then an error will be raised. If any member of *tlist* fails on a particular subgoal, then that subgoal is returned unchanged. *THEN_LIST* is an alias for *THEN_LIST_T*.

Errors
```
9101 number of tactics must equal the number of subgoals
```

As failures of the initial tactic.

SML
```
val THEN_TRY_TTCL : (THM_TACTICAL * THM_TACTICAL) ->
        THM_TACTICAL;
```

Description *THEN_TRY_TTCL* is a theorem tactical combinator. It is an infix operator which applies the first theorem tactical, and then, if it succeeds, the second theorem tactical, using only the first result if the second fails.

Uses For use in programming theorem tacticals.

SML
```
val THEN_TRY_T : (TACTIC * TACTIC) -> TACTIC;
val THEN_TRY : (TACTIC * TACTIC) -> TACTIC;
```

Description *THEN_TRY_T* is a tactical used as an infix operator. *tac1 THEN_TRY_T tac2* is the tactic that applies *tac1* and then attempts to apply *tac2* to each resulting subgoal (perhaps none). If *tac2* fails on any particular subgoal then that subgoal will be unchanged from the result of *tac1*. If *tac1* fails then the overall tactic fails. *THEN_TRY* is an alias for *THEN_TRY_T*.

Errors As the errors of *tac1*.

SML
```
val THEN_TTCL : (THM_TACTICAL * THM_TACTICAL) -> THM_TACTICAL;
```

Description *THEN_TTCL* is a theorem tactical combinator. It is an infix operator which composes two theorem tacticals using ordinary function composition:

```
(tcl1 THEN_TTCL tcl2) thmtac thm = (tcl1 o tcl2) thmtac thm
```

Uses For use in programming theorem tacticals.

SML

```
val THEN_T : (TACTIC * TACTIC) -> TACTIC;
val THEN : (TACTIC * TACTIC) -> TACTIC;
```

Description *THEN_T* is a tactical used as an infix operator. *tac1 THEN_T tac2* is the tactic that applies *tac1* and then applies *tac2* to each resulting subgoal (perhaps none). *THEN* is an alias for *THEN_T*.

Errors As the errors of *tac1* and *tac2*.

SML

```
val TOP_ASM_T : (THM -> TACTIC) -> TACTIC;
```

Description If the top entry in the assumption list is *asm* say, *TOP_ASM_T thmtac* acts as *thmtac*(*asm*⊢*asm*).

Tactic

$$\frac{\{\ asm,\ \Gamma\ \}\ t}{thmtac\ (asm \vdash asm)\ (\{\ asm,\ \Gamma\ \}\ t)}$$

TOP_ASM_T
⌜*asm*⌝
thmtac

Uses To use an assumption as a theorem

Errors

9302	*the assumption list is empty*

SML

```
val TRY_TTCL : THM_TACTICAL -> THM_TACTICAL;
```

Description *TRY_TTCL ttcl* is a theorem tactical which applies *ttcl* if it can, and otherwise acts as *ID_THEN*.

Uses For use in programming theorem tacticals.

SML

```
val TRY_T : TACTIC -> TACTIC;
val TRY : TACTIC -> TACTIC;
```

Description *TRY_T tac* is a tactic which applies *tac* to the goal and if that fails leaves the goal unchanged. It is the same as *tac ORELSE id_tac*. *TRY* is an alias for *TRY_T*.

SML

```
val t_tac : TACTIC;
```

Description Prove a goal with conclusion '*T*'.

Tactic

$$\frac{\{\ \Gamma\ \}\ T}{}$$

t_tac

See Also *strip_tac*, *taut_tac*.

Uses Tactic programming.

Errors

28011	*Goal does not have the form* $\{\Gamma\}T$

SML
```
val var_elim_asm_tac : TERM -> TACTIC;
val var_elim_nth_asm_tac : int -> TACTIC;
val VAR_ELIM_ASM_T : TERM -> (THM -> TACTIC) -> TACTIC;
val VAR_ELIM_NTH_ASM_T : int -> (THM -> TACTIC) -> TACTIC;
```

Description These tactics and tacticals do variable elimination with a chosen assumption of the goal. They take an assumption of the form: ⌜*var* = *value*⌝ or ⌜*value* = *var*⌝, where *var* is a variable and, if the subterm *value* does not contain *var* free, they substitute *value* for the free variable *var* throughout the goal (discarding the original assumption).

If an assumption is an equation of variables, then the tactic will strip digits and the current variant suffix from the right of the two variable names, and will choose to eliminate the variable with the shortest remaining name string, taking eliminating the left hand side variable if the strings are of equal length (this is a heuristic). If the variables are the same then the assumption is just discarded with no further effect.

var_elim_asm_tac will determine whether its term argument is an assumption of the above form. If so, it will substitute for the free variable *var* with *value* throughout the goal, stripping any changed assumptions back into the goal (returning the rest by *check_asm_tac*), and then discard the original assumption. *VAR_ELIM_ASM_T* allows the users choice of function to be applied to the modified assumptions.

var_elim_nth_asm_tac works as *var_elim_asm_tac*, except it takes an integer indicating the "nth" assumption is to be used. *VAR_ELIM_NTH_ASM_T* allows the users choice of function to be applied to the modified assumptions.

See Also *all_var_elim_asm_tac1* and its kin to apply this sort of functionality to all the assumptions simultaneously. *prop_eq_prove_tac* for more sophisticated approach to these kinds of problems.

Errors

9301	*the term ?0 is not in the assumption list*
9303	*the index ?0 is out of range*
29027	*?0 is not of the form ⌜var = ...⌝ or ⌜... = var⌝ where the variable ⌜var⌝ is not free in ⌜...⌝*

SML
```
val ⇔_T2 : (THM -> TACTIC) -> (THM -> TACTIC) -> TACTIC;
```

Description Reduce a bi-implication by passing the operands to tactic generating functions.

Tactic

$$\frac{\{\ \Gamma\ \}\ t1 \Leftrightarrow t2}{ttac1\{\ t1,\ \Gamma\ \} \vdash t2;\ ttac2\{\ t2,\ \Gamma\ \} \vdash t1} \qquad \begin{array}{l} \Leftrightarrow\_T2 \\ ttac1\ \ ttac2 \end{array}$$

See Also *⇔_T*, *STRIP_CONCL_T*

Errors

28061	*Goal is not of the form: { Γ } t1 ⇔ t2*

SML

val $\Leftrightarrow$**_tac** : $TACTIC$;

Description Reduce a bi-implication to two subgoals.

Tactic

$$\frac{\{\ \Gamma\ \}\ t1 \Leftrightarrow t2}{\{strip\ t1,\ \Gamma\ \} \vdash t2;\ \{strip\ t2,\ \Gamma\ \} \vdash t1} \qquad \Leftrightarrow\_tac$$

See Also $strip\_tac$, $\Leftrightarrow\_T$

Errors

28061 Goal is not of the form: $\{\ \Gamma\ \}\ t1 \Leftrightarrow t2$

SML

val $\Leftrightarrow$**_THEN2** : (THM −> $TACTIC$) −> (THM −> $TACTIC$) −>
(THM −> $TACTIC$);

Description A theorem tactical to apply given theorem tactics to the the result of eliminating $\Leftrightarrow$ from a theorem of the form $\Gamma \vdash t1 \Leftrightarrow t2$.

$\Leftrightarrow\_THEN2\ ttac1\ ttac2(\Gamma \vdash t1 \Leftrightarrow t2) = ttac1(\Gamma \vdash t1 \Rightarrow t2)\ THEN\ ttac2(\Gamma \vdash t2 \Rightarrow t1)$

The function is partially evaluated with only the theorem tactic and theorem arguments.

See Also $\Leftrightarrow\_THEN$, $STRIP\_THM\_THEN$

Errors

28062 ?0 is not of the form: ‘$\Gamma \vdash t1 \Leftrightarrow t2$‘

SML

val $\Leftrightarrow$**_THEN** : (THM −> $TACTIC$) −> (THM −> $TACTIC$);

Description A theorem tactical to apply a given theorem tactic to the result of eliminating $\Leftrightarrow$ from a theorem of the form $\Gamma \vdash t1 \Leftrightarrow t2$.

$\Leftrightarrow\_THEN\ thmtac\ (\Gamma \vdash t1 \Leftrightarrow t2) = thmtac\ (\Gamma \vdash t1 \Rightarrow t2)\ THEN\ thmtac\ (\Gamma \vdash t2 \Rightarrow t1)$

The function is partially evaluated with only the theorem tactic and theorem arguments.

See Also $\Leftrightarrow\_THEN2$, $STRIP\_THM\_THEN$

Errors

28062 ?0 is not of the form: ‘$\Gamma \vdash t1 \Leftrightarrow t2$‘

SML

val $\Leftrightarrow$**_t_tac** : $TACTIC$;

Description Simplifies a goal of the form: ...$\Leftrightarrow T$ or $T\Leftrightarrow$....

Tactic

$$\frac{\{\ \Gamma\ \}\ t \Leftrightarrow T}{\{\ \Gamma\ \}\ t} \qquad \Leftrightarrow\_t\_tac$$

and

Tactic

$$\frac{\{\ \Gamma\ \}\ T \Leftrightarrow t}{\{\ \Gamma\ \}\ t} \qquad \Leftrightarrow\_t\_tac$$

Errors

28012 Goal not of form: $\{\ \Gamma\ \}\ t \Leftrightarrow T$ *or* $\{\ \Gamma\ \}\ T \Leftrightarrow t$

See Also $strip\_tac$

Uses Tactic programming.

SML
```
val ⇔_T : (THM -> TACTIC) -> TACTIC;
```

Description Reduce a bi-implication by passing each operand to a tactic generating function.

Tactic

$$\frac{\{\ \Gamma\ \}\ t1 \Leftrightarrow t2}{ttac\{\ t1,\ \Gamma\ \} \vdash t2;\ ttac\{\ t2,\ \Gamma\ \} \vdash t1} \qquad \begin{array}{l} \Leftrightarrow\_T \\ ttac \end{array}$$

See Also *⇔_T2*, *STRIP_CONCL_T*

Errors
```
28061 Goal is not of the form: { Γ } t1 ⇔ t2
```

SML
```
val ∧_tac : TACTIC;
```

Description Reduce the proof of a conjunction to the proof of its conjuncts.

Tactic

$$\frac{\{\ \Gamma\ \}\ t1 \land t2}{\{\ \Gamma\ \}\ t1;\ \{\ \Gamma\ \}\ t2} \qquad \land\_tac$$

See Also *strip_tac*

Errors
```
28031 Goal is not of the form: { Γ } t1 ∧ t2
```

SML
```
val ∧_THEN2 : (THM -> TACTIC) -> (THM -> TACTIC) -> (THM -> TACTIC);
```

Description A theorem tactical to apply given theorem tactics to the conjuncts of a theorem of the form $\Gamma \vdash t1 \land t2$.

```
∧_THEN2 thmtac1 thmtac2 (Γ ⊢ t1 ∧ t2) = thmtac1 (Γ ⊢ t1) THEN thmtac2 (Γ ⊢ t2)
```

See Also *∧_THEN*, *STRIP_THM_THEN*

Errors
```
28032 ?0 is not of the form: ‘Γ ⊢ t1 ∧ t2‘
```

SML
```
val ∧_THEN : (THM -> TACTIC) -> (THM -> TACTIC);
```

Description A theorem tactical to apply a given theorem tactic to the conjuncts of a theorem of the form $\Gamma \vdash t1 \land t2$.

```
∧_THEN thmtac (Γ ⊢ t1 ∧ t2) = thmtac (Γ ⊢ t1) THEN thmtac (Γ ⊢ t2)
```

The function may be partially evaluated with only its theorem tactic and theorem arguments.

See Also *∧_THEN2*, *STRIP_THM_THEN*

Errors
```
28032 ?0 is not of the form: ‘Γ ⊢ t1 ∧ t2‘
```

SML

$val\ \vee$_**left_tac** : *TACTIC*;

Description Take the left disjunct of the current goal as the subgoal.

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

Tactic

$$\frac{\{\ \Gamma\ \}\ a \vee b}{\{\ \Gamma\ \}\ a} \qquad \vee\_left\_tac$$

See Also $\vee$_*left_tac*, *swap_*$\vee$*_tac*, *strip_tac*

Errors

28041 Goal is not of the form: $\{\ \Gamma\ \}\ a \vee b$

SML

$val\ \vee$_**right_tac** : *TACTIC*;

Description Take the right disjunct of the current subgoal as the new subgoal.

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

Tactic

$$\frac{\{\ \Gamma\ \}\ a \vee b}{\{\ \Gamma\ \}\ b} \qquad \vee\_right\_tac$$

See Also $\vee$_*right_tac*, *swap_*$\vee$*_tac*, *strip_tac*

Errors

28041 Goal is not of the form: $\{\ \Gamma\ \}\ a \vee b$

SML

$val\ \vee$_**THEN2** : (*THM* –> *TACTIC*) –> (*THM* –> *TACTIC*) –> (*THM* –> *TACTIC*);

Description A theorem tactical to perform a case split on a given disjunctive theorem applying tactic generating functions to the extra assumption in each branch.

$\vee\_THEN2\ ttac1\ ttac2\ (\Delta \vdash t1 \vee t2)\ (\{\Gamma\}\ t) =$
$\quad ttac1\ (t1 \vdash t1)\ (\{\Gamma\}\ t);\ ttac2\ (t2 \vdash t2)(\{\Gamma\}\ t)$

The function may be partially evaluated with only its theorem tactic and theorem arguments.

See Also *STRIP_THM_THEN*, $\vee$_*THEN*

Errors

28042 ?0 is not of the form: ‘$\Gamma \vdash t1 \vee t2$‘

SML

$val\ \vee$_**THEN** : (*THM* –> *TACTIC*) –> (*THM* –> *TACTIC*);

Description A theorem tactical to perform a case split on a given disjunctive theorem applying a tactic generating function to the extra assumption in each branch.

$\vee\_THEN\ ttac\ (\Delta \vdash t1 \vee t2)\ (\{\Gamma\}\ t) = ttac\ (t1 \vdash t1)\ (\{\Gamma\}\ t);\ ttac\ (t2 \vdash t2)(\{\Gamma\}\ t)$

The function may be partially evaluated with only its theorem tactic and theorem arguments.

See Also *STRIP_THM_THEN*, $\vee$_*THEN2*

Errors

28042 ?0 is not of the form: ‘$\Gamma \vdash t1 \vee t2$‘

SML

val **¬_elim_tac** : $TERM$ −> $TACTIC$;

Description Proof by showing assumptions give rise to two contradictory subgoals.

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

Tactic

$$\frac{\{\ \Gamma\ \}\ t2}{\{\ \Gamma\ \}\ t1;\ \{\ \Gamma\ \}\ \neg\ t1} \qquad \begin{array}{l} \neg\_elim\_tac \\ \ulcorner t1 \urcorner \end{array}$$

The function may be partially evaluated with only its term argument.

Uses In tactic programming. If an assumption has its negation also in the assumption list this will make for a rapid proof. *asm_ante_tac t1 THEN strip_tac* is a more memorable idiom for handling such a case in interactive use but is a little slower.

See Also *strip_tac*

Errors

28022 ?0 is not boolean

SML

val **¬_in_conv** : $CONV$;

Description This is a conversion which moves a top level negation inside other predicate calculus connectives using whichever one of the following rules applies:

$$\begin{array}{lcl}
\neg\neg t & = & t \\
\neg(t1\ \wedge\ t2) & = & \neg t1\ \vee\ \neg t2 \\
\neg(t1\ \vee\ t2) & = & \neg t1\ \wedge\ \neg t2 \\
\neg(t1\ \Rightarrow\ t2) & = & t1\ \wedge\ \neg t2 \\
\neg(t1\ \Leftrightarrow\ t2) & = & (t1\ \wedge\ \neg t2)\ \vee\ (t2\ \wedge\ \neg t1) \\
\neg(if\ \ a\ \ then\ \ t1\ \ else\ \ t2) & & \\
 & = & (if\ \ a\ \ then\ \neg t1\ \ else\ \neg t2) \\
\neg\forall vs \bullet t & = & \exists vs \bullet \neg t \\
\neg\exists vs \bullet t & = & \forall vs \bullet \neg t \\
\neg\exists_1 vs \bullet t & = & \forall vs \bullet \neg(t\ \wedge\ \forall vs' \bullet t[vs']\ \Rightarrow\ vs'\ =\ vs) \\
\neg T & = & F \\
\neg F & = & T
\end{array}$$

Uses Tactic and conversion programming.

See Also *simple_¬_in_conv*, *¬_in_tac*

Errors

28131 No applicable rules for the term ?0

SML
```
val ¬_in_tac : TACTIC;
```

Description This is a tactic which moves a top level negation in the conclusion of the goal inside other predicate calculus connectives using the following rules:

$$\begin{array}{lcl}
\neg\neg t & \rightarrow & t \\
\neg(t1 \wedge t2) & \rightarrow & \neg t1 \vee \neg t2 \\
\neg(t1 \vee t2) & \rightarrow & \neg t1 \wedge \neg t2 \\
\neg(t1 \Rightarrow t2) & \rightarrow & t1 \wedge \neg t2 \\
\neg(t1 \Leftrightarrow t2) & \rightarrow & (t1 \wedge \neg t2) \vee (t2 \wedge \neg t1) \\
\neg\forall vs\bullet t & \rightarrow & \exists vs\bullet\neg t \\
\neg\exists vs\bullet t & \rightarrow & \forall vs\bullet\neg t \\
\neg\exists_1 vs\bullet t & \rightarrow & \forall vs\bullet\neg(t \wedge \forall vs'\bullet t[vs'] \Rightarrow vs' = vs) \\
\neg T & \rightarrow & F \\
\neg F & \rightarrow & T
\end{array}$$

Uses

See Also *simple_¬_in_tac*, *¬_in_conv*

Errors
```
28025 No applicable rule for this goal
```

SML
```
val ¬_IN_THEN : THM_TACTICAL;
```

Description This is a theorem tactical which applies a given theorem tactic to the result of transforming a theorem by moving a top level negation inside other predicate calculus connectives using the following rules:

$$\begin{array}{lcl}
\neg\neg t & \rightarrow & t \\
\neg(t1 \wedge t2) & \rightarrow & \neg t1 \vee \neg t2 \\
\neg(t1 \vee t2) & \rightarrow & \neg t1 \wedge \neg t2 \\
\neg(t1 \Rightarrow t2) & \rightarrow & t1 \wedge \neg t2 \\
\neg(t1 \Leftrightarrow t2) & \rightarrow & (t1 \wedge \neg t2) \vee (t2 \wedge \neg t1) \\
\neg\forall vs\bullet t & \rightarrow & \exists vs\bullet\neg t \\
\neg\exists vs\bullet t & \rightarrow & \forall vs\bullet\neg t \\
\neg\exists_1 vs\bullet t & \rightarrow & \forall vs\bullet\neg(t \wedge \forall vs'\bullet t[vs'] \Rightarrow vs' = vs) \\
\neg T & \rightarrow & F \\
\neg F & \rightarrow & T
\end{array}$$

This function partially evaluates given only the theorem and theorem-tactical.

See Also *SIMPLE_¬_IN_THEN*

Errors
```
29010 No applicable rule for ?0
```

SML

val $\neg$**_rewrite_canon** : THM $->$ THM $list$
val $\forall$**_rewrite_canon** : THM $->$ THM $list$

Description These are some of the standard canonicalisation functions used for breaking theorems up into lists of equations for use in rewriting. They four perform the following transformations:

$\neg$*_rewrite_canon*	$(\Gamma \vdash \neg(t1 \lor t2))$	$= (\Gamma \vdash \neg t1 \land \neg t2)$
$\neg$*_rewrite_canon*	$(\Gamma \vdash \neg\exists vs\bullet t)$	$= (\Gamma \vdash \forall vs\bullet\neg t)$
$\neg$*_rewrite_canon*	$(\Gamma \vdash \neg\neg t)$	$= (\Gamma \vdash t)$
$\neg$*_rewrite_canon*	$(\Gamma \vdash \neg t)$	$= (\Gamma \vdash t \Leftrightarrow F)$
$\forall$*_rewrite_canon*	$(\Gamma \vdash \forall vs\bullet t)$	$= \Gamma \vdash t$

See Also *simple_¬_rewrite_canon*, *simple_∀_rewrite_canon*.

Errors

26201 Failed as requested

The area given by the failure will be *fail_canon*.

SML

val $\neg$**_T2** : $TERM$ $->$ $(THM$ $->$ $TACTIC)$ $->$ $(THM$ $->$ $TACTIC)$ $->$ $TACTIC$;

Description A form of proof by contradiction using two theorem tactics to simplify the subgoals.

Note that *strip_tac* may be used to push a negation inside other logical connectives, which is often the best way of handling a negated goal.

Tactic

$$\frac{\{\ \Gamma\ \}\ \neg t2}{\begin{array}{c} ttac1\ (t2 \vdash t2)\ \{\ \Gamma\ \}\ t1; \\ ttac2\ (t2 \vdash t2)\ \{\ \Gamma\ \}\ \neg\ t1 \end{array}} \qquad \begin{array}{l} \neg\_T2 \\ ttac1\ \ ttac2 \\ \ulcorner t1 \urcorner \end{array}$$

Uses To prove a negated term by showing that assuming the term gives rise to a contradiction.

See Also *strip_tac*, *contr_tac*, *¬_tac*, *STRIP_CONCL_T*, *¬_in_conv*

Errors

28022 ?0 is not boolean
28023 Goal is not of the form $\ulcorner\neg\ t\urcorner$

SML

```
val ¬_tac : TERM -> TACTIC;
```

Description A form of proof by contradiction as a tactic: $\neg t2$ holds if $t2 \vdash t1$ and $t2 \vdash \neg t1$ for some term $t1$.

Note that *strip_tac* may be used to push a negation inside other logical connectives, which is often the best way of handling a negated goal.

Tactic

$$\frac{\{\ \Gamma\ \}\ \neg t2}{\{strip\ t2,\ \Gamma\ \}\ t1;\ \{strip\ t2,\ \Gamma\ \}\ \neg\ t1} \qquad \neg\_tac \quad \ulcorner t1 \urcorner$$

Uses To prove a negated term by showing that assuming the term gives rise to a contradiction.

See Also *strip_tac*, *contr_tac*, *¬_T*

Errors

28022 ?0 is not boolean
28023 Goal is not of the form ⌜¬ t⌝

SML

```
val ¬_T : TERM -> (THM -> TACTIC) -> TACTIC;
```

Description A form of proof by contradiction using a theorem tactic to simplify the subgoals.

Note that *strip_tac* may be used to push a negation inside other logical connectives, which is often the best way of handling a negated goal.

Tactic

$$\frac{\{\ \Gamma\ \}\ \neg t2}{\begin{array}{c} ttac\ (t2 \vdash t2)\ \{\ \Gamma\ \}\ t1; \\ ttac\ (t2 \vdash t2)\ \{\ \Gamma\ \}\ \neg\ t1 \end{array}} \qquad \begin{array}{l} \neg\_T \\ ttac \\ \ulcorner t1 \urcorner \end{array}$$

Uses To prove a negated term by showing that assuming the term gives rise to a contradiction.

See Also *strip_tac*, *contr_tac*, *¬_tac*, *STRIP_CONCL_T*, *¬_in_conv*

Errors

28022 ?0 is not boolean
28023 Goal is not of the form ⌜¬ t⌝

SML

```
val ¬_¬_thm : THM
val ¬_∨_thm : THM
val ¬_∧_thm : THM
val ¬_⇒_thm : THM
val ¬_⇔_thm : THM
val ¬_if_thm : THM
val ¬_f_thm : THM

val ⇒_thm : THM
val ⇔_thm : THM
val if_thm : THM
```

Description These theorems are tautologies saved in the theory "misc" because they are frequently used in tactic and conversion programming.

The first seven theorems are De Morgan's laws for the various propositional connectives formulated so that they can be used to normalise a propositional term by moving all negations inside other connectives. *¬_t_thm* is also provided but is documented elsewhere.

The last three theorems give definitions for implication, bi-implication and conditional in terms of disjunction, conjunction and negation.

```
¬_¬_thm        ⊢ ∀a•¬ ¬a ⇔ a
¬_∨_thm        ⊢ ∀a b•¬ (a ∨ b) ⇔ (¬a ∧ ¬b)
¬_∧_thm        ⊢ ( ¬(a ∧ b) ⇔ (¬a ∨ ¬b)
¬_⇒_thm        ⊢ ∀a b•¬(a ⇒ b) ⇔ (a ∧ ¬b)
¬_⇔_thm        ⊢ ∀ a b • ¬ (a ⇔ b) ⇔ a ∧ ¬ b ∨ b ∧ ¬ a
¬_if_thm       ⊢ ∀ a b • ¬ (if a then T else T) ⇔ (if a then ¬ T else ¬ T)

¬_f_thm        ⊢ ¬F ⇔ T
⇒_thm          ⊢ ∀a b•(a ⇒ b) ⇔ (¬a ∨ b)
⇔_thm          ⊢ ∀a b•(a ⇔ b) ⇔ (a ⇒ b) ∧ (b ⇒ a)
if_thm         ⊢ ∀a b c•(if a then b else c) ⇔ (a ∧ b) ∨ (¬a ∧ c)
```

See Also *¬_t_thm*.

SML

```
val ⇒_tac : TACTIC;
```

Description Strip the antecedent of an implicative goal into the assumption list.

Tactic

$$\frac{\{\ \Gamma\ \}\ t1\ \Rightarrow\ t2}{\{strip\ t1,\ \Gamma\ \}\ t2} \qquad \Rightarrow\_tac$$

Errors

```
28051 Goal is not of form: { Γ } t1 ⇒ t2
```

SML

```
val ⇒_THEN : (THM -> TACTIC) -> (THM -> TACTIC);
```

Description A theorem tactical to apply a given theorem tactic to the result of eliminating ⇒ from a theorem of the form $\Gamma \vdash t1 \Rightarrow t2$.

$$\Rightarrow\_THEN\ thmtac\ (\Gamma \vdash t1 \Rightarrow t2) = thmtac\ (\Gamma \vdash \neg\ t1 \lor t2)$$

The function is partially evaluated with only the theorem tactic and theorem arguments.

Errors

28054 ?0 is not of the form: ‘Γ ⊢ t1 ⇒ t2‘

SML

```
val ⇒_thm_tac : THM -> TACTIC;
```

Description A tactic which uses a theorem whose conclusion is an implication, $t1 \Rightarrow t2$, to reduce a goal with conclusion $t2$ to $t1$.

Tactic

$$\frac{\{\ \Gamma\ \}\ t2}{\{\ \Gamma\ \}\ t1} \qquad \begin{array}{l} \Rightarrow\_thm\_tac \\ \Gamma 1 \vdash t1 \Rightarrow t2 \end{array}$$

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

Uses Mainly for use in tactic programming where the extra generality of *bc_thm_tac* and *bc_tac* is not required.

See Also *bc_thm_tac*, *bc_tac*.

Errors

29013 Conclusion of the goal is not ?0

SML

```
val ⇒_T : (THM -> TACTIC) -> TACTIC;
```

Description Reduce an implicative goal by passing the antecedent to a tactic generating function.

Tactic

$$\frac{\{\ \Gamma\ \}\ t1 \Rightarrow t2}{ttac\{\ t1,\ \Gamma\ \}\ t2} \qquad \begin{array}{l} \Rightarrow\_T \\ ttac \end{array}$$

Errors

28051 Goal is not of form: { Γ } t1 ⇒ t2

SML

```
val ∀_tac : TACTIC;
```

Description Reduce a universally quantified goal.

Tactic

$$\frac{\{\ \Gamma\ \}\ \forall\ vs[x1,...] \bullet t[x1,...]}{\{\ \Gamma\ \}\ t[x1',...]} \qquad \forall\_tac$$

where $x1'$ is a variant name of $x1$, etc, different from any variable in Γ or t.

See Also *simple_∀_tac*

Errors

29001 Goal is not of the form: { Γ } ∀ vs • t[vs]

SML
```
val ∃_tac : TERM -> TACTIC ;
```

Description Provide a witness for an existential subgoal.

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

Tactic

$$\frac{\{\ \Gamma\ \}\ \exists\ vs[x1,...] \bullet t2[x1,...]}{\{\ \Gamma\ \}\ t2[t1',...]} \qquad \begin{array}{l}\exists\_tac \\ \ulcorner t \urcorner\end{array}$$

where $vs[t\_1,...]$ is t, type instantiated to have the same type as $vs[x1,...]$, and broken up using *Fst* and *Snd* as necessary.

See Also *simple_∃_tac*

Errors
```
29002 Goal is not of the form: { Γ } ∃ vs • t2[vs]
29008 Cannot match witness ?0 to varstruct ?1
```

SML
```
val ∃_THEN : (THM -> TACTIC) -> (THM -> TACTIC);
```

Description A theorem tactical which applies a given theorem tactic to the result of eliminating the outermost quantifier from a theorem of the form $\Gamma \vdash \exists vs \bullet t$.

$$\exists\_THEN\ thmtac\ (\Gamma \vdash \exists vs[x1,...] \bullet t) = thmtac\ (\Gamma \vdash t[x1'/x1,...])$$

where ⌜x1'⌝ is a variant of ⌜x1⌝ , etc, which does not appear in Γ or in the assumption or conclusion of the goal.

See Also *SIMPLE_∃_THEN*

Errors
```
29003 ?0 is not of the form: ‘Γ ⊢ ∃ vs • t‘
```

SML
```
val ∃₁_tac : TERM -> TACTIC;
```

Description Provide a witness for a goal with conclusion of the form $\exists_1 x \bullet t$.

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

Tactic

$$\frac{\{\ \Gamma\ \}\ \exists_1\ vs[x1,...] \bullet P[x1,...]}{\begin{array}{c}\{\ \Gamma\ \}\ P[t1',...]; \\ \{\ \Gamma\ \}\ \forall vs[x1',...] \bullet \\ P[x1',...] \Rightarrow vs[x1',...] = t'\end{array}} \qquad \begin{array}{l}\exists_1\_tac1 \\ \ulcorner t \urcorner\end{array}$$

where x_i' is a variant of x_i which does not occur free in t, t' is equal to t type instantiated to the type of $vs[x1,...]$, and $vs[t1',...]$ equals t' (perhaps using *Fst* and *Snd*).

Errors
```
29004 Goal is not of the form: { Γ } ∃₁ vs • t
29008 Cannot match witness ?0 to varstruct ?1
```

SML
```
val ∃₁_THEN : (THM -> TACTIC) -> (THM -> TACTIC);
```

Description A theorem tactical which applies a given theorem tactic to the result of eliminating the outermost quantifier from a theorem of the form $\Gamma \vdash \exists\_1 vs \bullet t$.

$$\begin{aligned}&\exists_1\_THEN\ thmtac\ (\Gamma \vdash \exists_1 vs[x1,...]\bullet t) = \\ &\quad thmtac\ (\Gamma \vdash t[x1'/x1,...] \land \\ &\quad\quad \forall vs[x1'',...]\bullet P[x1'',...] \Rightarrow vs[x1'',...] = vs[x1',...])\end{aligned}$$

where ⌜x1'⌝ and ⌜x1''⌝ are distinct variants of ⌜x1⌝, etc, which do not appear free in Γ or in the assumptions or conclusion of the goal.

Errors
29005 ?0 is not of the form: '$\Gamma \vdash \exists_1\ vs \bullet t$'

SML
```
val ε_tac : TERM -> TACTIC;
val ε_T : TERM -> (THM -> TACTIC) -> TACTIC;
```

Description Given a choice term, $\epsilon x \bullet t$ say, *ε_tac* sets $\exists x \bullet t$ as a lemma, and derives the new assumption $t[\epsilon x \bullet t/x]$ from it.

ε_T is the same as *ε_tac* except that it passes the new assumption to a tactic generating function.

Tactic

$$\frac{\{\ \Gamma\ \}\ t1}{\{\ \Gamma\ \}\ \exists x \bullet t;\ \{strip\ t[\epsilon x \bullet t/x],\ \Gamma\ \}\ t1} \qquad \begin{array}{l}\epsilon\_tac \\ \ulcorner \epsilon x \bullet t \urcorner\end{array}$$

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable. This occurs when the use of the choice function is in some sense irrelevant to the truth of the goal, e.g., $(\epsilon x \bullet T) = (\epsilon x \bullet T)$.

See Also *all_ε_tac*, *all_ε_T* (which are easier to use in most cases).

Errors
29050 ?0 is not of the form ⌜$\epsilon x \bullet p\ x$⌝

8.4 Propositional Equational Reasoning

SML

| *signature* **PropositionalEquality** = *sig*

Description This is the signature of a structure containing proof procedures for propositional calculus with equality.

SML

| (∗ *Proof Context*: **prop_eq** ∗)

Description This is a complete proof context whose purpose is to supply a decision procedure for problems involving sets of equalities and the propositional calculus.

Contents The rewriting, theorem stripping and conclusion stripping components are as for the proof context *predicates* (q.v.). The automatic proof tactic is *prop_eq_prove_tac* (q.v.) The automatic proof conversion just tries to prove its argument, t say, using the automatic proof tactic and returns $t \Leftrightarrow T$ if it succeeds.

SML

| (∗ *Proof Context*: **′prop_eq** ∗)

Description This is a component proof context whose purpose is to supply a decision procedure for problems involving sets of equalities and the propositional calculus.

Contents The automatic proof components are as for proof context *prop_eq*. Other components are blank.

SML

| (∗ *Proof Context*: **prop_eq_pair** ∗)

Description This is a complete proof context whose main purpose is to supply a decision procedure for problems involving sets of equalities, the propositional calculus and pairing.

Contents The rewriting, theorem stripping and conclusion stripping components are as for the proof context *predicates* (q.v.) each augmented with conversion *pair_eq_conv* (q.v.) which effect the following transformations:

$$
\begin{array}{lcl}
Fst(a,b) = x & \to & a = x \\
Snd(a,b) = y & \to & b = y \\
x = Fst(a,b) & \to & x = a \\
y = Snd(a,b) & \to & y = b \\
(a,b) = (c,d) & \to & a = c \land b = d \\
(a,b) = z & \to & a = Fst\ z \land b = Snd\ z \\
z = (a,b) & \to & Fst\ z = a \land Snd\ z = b \\
z = w & \to & Fst\ z = Fst\ w \land Snd\ z = Snd\ w
\end{array}
$$

The automatic proof tactic is *prop_eq_prove_tac* (q.v.). The automatic proof conversion just tries to prove its argument, t say, using the automatic proof tactic and returns $t \Leftrightarrow T$ if it succeeds.

SML

| (∗ *Proof Context*: **′prop_eq_pair** ∗)

Description This is a component proof context whose purpose is to supply a decision procedure for problems involving sets of equalities, the propositional calculus and pairing.

Contents The rewriting, theorem stripping and conclusion stripping components contain only the *pair_eq_conv* conversion. The automatic proof components are as for *prop_eq_pair*. Other components are blank.

SML
```
val ASM_PROP_EQ_T : (THM list -> TACTIC) -> THM list -> TACTIC
val PROP_EQ_T : (THM list -> TACTIC) -> THM list -> TACTIC
```

Description These are theorem tacticals which process the argument theorems and (for *ASM_PROP_EQ_T*) the assumptions before calling the argument theorem tactic. A call of "*ASM_PROP_EQ_T thm_tac thms*" takes *thms* plus theorems representing any equations from the assumptions, these are cannonicalised by the rewriting canon of the current proof context, then processed by *prop_eq_rule* (q.v.) to form the arguments passed to function *thm_tac*. The order of the assumptions may be changed. Tactical *PROP_EQ_T* does not use the assumptions.

Uses With the rewriting tactics.

SML
```
val pair_eq_conv : CONV
```

Description This conversion transforms equations involving pairs and the constants *Fst* and *Snd* into new equations whose comparands have simpler types by using the first match found in the following rules:

$Fst(a,b) = x$	$\rightarrow$	$a = x$
$Snd(a,b) = y$	$\rightarrow$	$b = y$
$x = Fst(a,b)$	$\rightarrow$	$x = a$
$y = Snd(a,b)$	$\rightarrow$	$y = b$
$(a,b) = (c,d)$	$\rightarrow$	$a = c \wedge b = d$
$(a,b) = z$	$\rightarrow$	$a = Fst\ z \wedge b = Snd\ z$
$z = (a,b)$	$\rightarrow$	$Fst\ z = a \wedge Snd\ z = b$
$z = w$	$\rightarrow$	$Fst\ z = Fst\ w \wedge Snd\ z = Snd\ w$

Uses The conversion is intended for use in tactic and conversion programming. It is usefully applied before using *prop_eq_prove_tac* or *ASM_PROP_EQ_T* (q.v.). The normal interactive interface is via rewriting or stripping in the proof context *prop_eq_pair* (q.v.).

Errors
```
84001 ?0 is not an equation involving pairs
```

SML

```
val prop_eq_prove_tac : THM list -> TACTIC;
```

Description This tactic is suitable to be used as an automatic proof procedure in a proof context, it aims to solve problems which may be solved by reasoning in the propositional calculus with equality.

The tactic has the following steps:

1. It strips all of the assumptions, using the stripping functions of the current proof context, back into the assumptions. More precisely, '*DROP_ASMS_T* (*MAP_EVERY* *strip_asm_tac*)' is used.
2. It applies *contr_tac* to increase the stock of assumptions.
3. It splits all of the assumptions into two groups, those which are equations and those which are not.
4. Using the equation assumptions and the given theorems, a new set of theorems is produced using *prop_eq_rule* (q.v.) which equate all members of an equivalence classes to a common member of the class.
5. It rewrites all of the other assumptions with these new theorems and with the rewriting theorems of the current proof context.
6. It strips the rewritten assumptions and the equational assumptions from step 3 back into the goal.

SML

```
val prop_eq_rule : THM list -> THM list * THM list;
```

Description Given a list of theorems with conclusions of the form ⌜ $a_i = b_i$ ⌝ for various a_i and b_i this function produces a set of theorems that equate all members of each equivalence class determined by the equations to a common value. The equivalence classes are the sets of all a_i and b_i that are equated either directly or transitively, they comprise terms that are α-convertible rather than requiring strict equality. For each of the equivalence classes a set of theorems equating each term in the class to the "simplest" (see below) term in the class is generated. These new theorems have the simplest term as their right hand comparand, duplicated theorems and identity theorems are excluded. The first list in the result tuple contains the new theorems from all of the equivalence classes. The second list in the result tuple comprises all the argument theorems which were not equasions. The new theorems are intended to be used as arguments for a rewriting operation.

The choice of the "simplest" term is intended to give the most useful rewriting theorems and those which are least likely to loop. HOL constants are considered the most simple, variables next, then functional applications, with lamdba abstractions considered the most complex. A simple recursive counting function is used to traverse each term to evaluate its complexity. Function *term_order* (q.v.) is used when the counting function cannot decide.

Example

Applying this rule to a list of theorems with the following conclusions:

⊢ $a1=b1$	⊢ $a1=c1$	⊢ $d1=c1$	⊢ $z1=x1$
⊢ $b1=y1$	⊢ $z1=w1$	⊢ $w1=y1$	⊢ $c1=y1$
⊢ $a2=b2$	⊢ $a2=c2$	⊢ $d2=c2$	⊢ $z2=x2$
⊢ $b2=y2$	⊢ $z2=w2$	⊢ $w2=y2$	⊢ $c2=y2$
⊢ $x \land y$			

will produce a list of theorems with the following conclusions as the first element of the result tuple:

⊢ $x1=a1$	⊢ $z1=a1$	⊢ $w1=a1$	⊢ $d1=a1$
⊢ $y1=a1$	⊢ $b1=a1$	⊢ $c1=a1$	
⊢ $x2=a2$	⊢ $z2=a2$	⊢ $w2=a2$	⊢ $d2=a2$
⊢ $y2=a2$	⊢ $b2=a2$	⊢ $c2=a2$	

plus the non equational theorems the second element of the result tuple.

8.5 Algebraic Normalisation

SML

```
signature Normalisation = sig
```

Description This is the signature of a structure containing conversions for monomial and polynomial term normalisation and related metalanguage functions.

SML

```
val anf_conv : CONV;
val ANF_C : CONV -> CONV;
```

Description *anf_conv* is a conversion which proves theorems of the form $\vdash t1 = t2$ where $t1$ is a term formed from atoms of type $\mathbb{N}$ and $t2$ is in what we may call additive normal form, i.e. it has the form: $t_1 + t_2 + ...$, where the t_i have the form $s_1 * s_2 * ...$where the s_i are atoms. Here, by atom we mean a term which is not of the form $t_1 + t_2 + ...$or $s_1 * s_2 * ...$.

The summands t_i and, within them, the factors s_j are given in increasing order with respect to the ordering on terms given by the function *term_order*, q.v. Arithmetic computation is carried out on atoms to ensure that at most one of the summands is a numeric literal and that, within each summand, at most one factor is a numeric literal. Any literal appears at the beginning of its factor or summand and addition of *0* or multiplication by *1* is simplified out.

ANF_C conv is a conversion which acts like *anf_conv* but which applies *conv* to each atom as it is encountered (and normalises the result recursively). The argument conversion may signal that it does not wish to change a subterm, t say, either by failing or by returning $t = t$, the former approach is more efficient.

The conversions fail with error number 81032 if there are no changes to be made to the term.

Errors

81032 ?0 is not of type ⌜:ℕ⌝ or is already in additive normal form

SML

```
val ASYM_C : CONV -> CONV
val GEN_ASYM_C : TERM ORDER -> CONV -> CONV
```

Description These conversionals allow one to control the behaviour of a conversion by making it asymmetric with respect to an ordering relation on terms (in the sense that the resulting conversion will only prove theorems of the form $t1 = t2$ in which $t2$ strictly precedes $t1$ in the ordering.

ASYM_C c is a conversion which behaves like c on terms $t1$ for which c $t1$ is a theorem with conclusion $t1 = t2$ where $t2$ (strictly) precedes $t1$ in the standard ordering on terms given by *term_order* q.v. and fails on other terms.

GEN_ASYM_C is like *ASYM_C* but allows the ordering function used to be supplied as a parameter. The parameter is interpreted as an ordering relation on terms in the same sense as the ordering relations used by *sort*, q.v.

Errors

81010 The conversion did not decrease the order of the term
81011 On argument ?0 the conversion returned ?1 which is not an equation

SML

```
val cnf_conv : CONV;
```

Description This is a conversion which proves theorems of the form $\vdash t1 \Leftrightarrow t2$ where $t2$ is in conjunctive normal form, i.e. either T or F or a conjunction of one or more disjunctions in which each disjunct is a propositional atom. Here, by atom we mean either a term whose principal connective is not a propositional calculus connective or the negation of such a term.

The conversion simplifies disjunctions and conjunctions as they are generated according to the following schema.

$$
\begin{array}{lcl}
a \wedge T & \to & a \\
T \wedge a & \to & a \\
F \wedge a & \to & F \\
a \wedge F & \to & F \\
a \wedge a & \to & a \\
a \wedge \neg a & \to & F \\
\\
a \vee T & \to & T \\
T \vee a & \to & T \\
F \vee a & \to & a \\
a \vee F & \to & a \\
a \vee a & \to & a \\
a \vee \neg a & \to & T \\
\\
\neg T & \to & F \\
\neg F & \to & T
\end{array}
$$

Note, however, that more global simplifications are not done, e.g. there is no attempt to eliminate a conjunct all of whose constituent atoms are contained in another conjunct. Thus, the conversion will not automatically prove tautologies.

The conversion fails with error number 81030 if there are no changes to be made to the term.

See Also *strip_tac* and *taut_rule* which supply a more useful and efficient means for working with the propositional calculus in most cases.

Errors

81030 ?0 is not of type ⌜:BOOL⌝ or is already in conjunctive normal form

SML

val **dnf_conv** : *CONV*;

Description This is a conversion which proves theorems of the form $\vdash t1 \Leftrightarrow t2$ where *t2* is in disjunctive normal form, i.e. either T or F or a disjunction of one or more conjunctions in which each conjunct is a propositional atom. Here, by atom we mean either a term whose principal connective is not a propositional calculus connective or the negation of such a term.

The conversion simplifies disjunctions and conjunctions as they are generated according to the following schema.

$$
\begin{array}{ll}
a \wedge T \rightarrow & a \\
T \wedge a \rightarrow & a \\
F \wedge a \rightarrow & F \\
a \wedge F \rightarrow & F \\
a \wedge a \rightarrow & a \\
a \wedge \neg a \rightarrow & F \\
 & \\
a \vee T \rightarrow & T \\
T \vee a \rightarrow & T \\
F \vee a \rightarrow & a \\
a \vee F \rightarrow & a \\
a \vee a \rightarrow & a \\
a \vee \neg a \rightarrow & T \\
 & \\
\neg T \rightarrow & F \\
\neg F \rightarrow & T
\end{array}
$$

Note, however, that more global simplifications are not done, e.g. there is no attempt to eliminate a disjunct all of whose constituent atoms are contained in another disjunct. Thus, the conversion will not automatically prove tautologies.

The conversion fails with error number 81031 if there are no changes to be made to the term.

See Also *strip_tac* and *taut_rule* which supply a more useful and efficient means for working with the propositional calculus in most cases.

Errors

81031 ?0 is not of type ⌜:BOOL⌝ or is already in disjunctive normal form

SML

val **gen_term_order** : (*TERM* −> (*TERM* ∗ *INTEGER*)) −> *TERM* −> *TERM* −> *int*;

Description *gen_term_order* gives a means of creating orderings on terms. It is retained for backwards compatibility, *make_term_order* now being the recommended way of constructing term orderings.

In the call *gen_term_order special*, the idea is that whenever two terms, *tm1* and *tm2* say, are compared, *special* is applied to them to produce two pairs, (*tm1′*, *k1*) and (*tm2′*, *k2*) say. These pairs are then compared lexicographically (using the ordering recursively for the first components, in a similar way to *term_order*, q.v.). It is the caller's responsibility to provide an argument *special* which will ensure that this procedure terminates. A sufficient condition is only to use functions *special* with the property that for some disjoint sets of terms *X_1*, *X_2*, ..., we have that *special tm* = (*tm*, *0*) if *tm* ∉ *X_i* for any *i* and that *special tm* = (*x_i*, *f_i*(*tm*)) if *tm* ∈ *X_i*, where *x_i* is a fixed element of *X_i* and *f_i* is a fixed injection of *X_i* into the natural numbers.

See Also *make_term_order1* which is now the recommended way of constructing new term orderings.

SML

val **make_term_order** :
(*TERM ORDER* −> *TERM ORDER*) *list* −> *TERM ORDER*;

Description *make_term_order* provides a systematic method for constructing term orderings. Its argument is a list of term order combinators: i.e., endofunctions on the type of term orderings.

The orderings *make_term_order* returns are derived from a base ordering on terms which works as follows:

1. Constants are ordered lexicographically by name (using *ascii_order*), then type (using *type_order*).
2. Variables are ordered lexicographically by name (using *ascii_order*), then type (using *type_order*).
3. Simple λ-abstractions are ordered lexicographically by recursion, bound variable first, then matrix.
4. Applications ordered lexicographically by recursion, function first, then operand.

If the above function ewre called *base*, then the ordering *make_term_order* [*f*, *g*, ..., *h*] acts as: *f*(*g*(...(*h*(*base*))...)) where each recursive call in *base* is a call on *make_term_order* [*f*, *g*, ..., *h*].

For example, the following defines an ordering on terms whcih makes the immediate successor of any term of type *BOOL* its immediate successor:

```
fun f t = (dest_¬ t, 1) handle Fail _ => (t, 0);

val ¬_order = make_term_order [fn r => induced_order(f, pair_order r int_order)];
```

SML

```
val poly_conv : TERM ORDER ->
        THM -> THM -> THM -> THM -> THM ->
        CONV -> CONV -> CONV -> CONV;
```

Description This conversion normalises terms constructed from atoms using two operators, both associative and commutative, the second of which, say op_* distributes over the other, say op_+. For clarity, we write the two operators with infix syntax although they need not actually be infix constants. Here, by "atom" we mean any term which is not of the form $t1\ op_+\ t2$ or $t1\ op_*\ t2$. The theorems computed by the conversion have the form $t = t_1\ op_+\ t_2\ op_+\ ...$, where the $t_i$are in non-decreasing order with respect to the ordering on terms given by the first parameter and have the form $s_1\ op_*\ s_2\ op_*\ ...$, where the s_i are atoms and are in non-decreasing order.

The associativity and commutativity of the operators and the distributivity are given as the five theorem parameters (which are also used to infer what the two operators are; n.b. the operators can be arbitrary terms, they need not be constants). The remaining parameters are conversions which are applied to each atom as it is encountered and to each subterm of the form $t_i\ op_+\ ...$ or $t_i\ op_*\ ...$ as it id created. In more detail the parameters are, in order, as follows:

1. A term ordering, such as *term_order*, q.v.
2. A theorem of the form $\vdash \forall x\ y \bullet x\ op_+\ y = y\ op_+\ x$.
3. A theorem of the form $\vdash \forall x\ y\ z \bullet (x\ op_+\ y)\ op_+\ z = x\ op_+\ y\ op_+\ z$.
4. A theorem of the form $\vdash \forall x\ y \bullet x\ op_*\ y = y\ op_*\ x$.
5. A theorem of the form $\vdash \forall x\ y\ z \bullet (x\ op_*\ y)\ op_*\ z = x\ op_*\ y\ op_*\ z$.
6. A theorem of the form $\vdash \forall x\ y\ z \bullet x\ op_*\ (y\ op_+\ z) = (x\ op_*\ y)\ op_+\ (x\ op_*\ z)$.
7. A conversion to be applied to any subterm of the form $t_i\ op_+\ ...$ whenever such a subterm is created. The result of the conversion will not be further normalised.
8. A conversion to be applied to any subterm of the form $t_i\ op_*\ ...$ whenever such a subterm is created. The result of the conversion will not be further normalised.
9. A conversion to be applied to any atom as it is encountered. If the conversion produces a non-atomic term, this is normalised recursively as it is produced.

The conversions supplied as parameters may signal that they do not wish to change a subterm, t say, either by failing or by returning $t = t$, the former approach is more efficient. The whole conversion fails with error number 81025 if there are no changes to be made to the term.

Errors

81023 *?0 does not have the form* $\vdash$ *t1 op1 (t2 op2 t3) = (t1 op1 t2) op2 (t1 op1 t3)*
81024 *?0 and ?1 do not have the forms* $\vdash$ *t1 op1 t2 = t2 op1 t1*
and $\vdash$ *t1 op1 (t2 op2 t3) = (t1 op1 t2) op2 (t1 op1 t3)*
81025 *?0 is already sorted*

SML

```
val sort_conv : TERM ORDER ->
        THM -> THM -> CONV -> CONV -> CONV;
```

Description This conversion normalises a term constructed from atoms using an associative and commutative binary operator, *op* say. For clarity, we write two operator with infix syntax although it need not actually be an infix constant. Here, by "atom" we mean any term which is not of the form *t1 op t2*. The theorems computed by the conversion have the form $t = t_1\ op\ t_2\ op\ ...$, where the t_i are in non-decreasing order with respect to the ordering on terms given by the first parameter.

The associativity and commutativity of the operator are given as the two theorem parameters (which are also used to infer what *op* is; n.b. *op* can be an arbitrary term, it need not be a constant). The remaining parameters are conversions which are applied to each atom as it is encountered and to each subterm of the form $t = t_i\ op\ ...$ as it is created. In more detail the parameters are, in order, as follows:

1. A term ordering, such as *term_order*, q.v.
2. A theorem of the form $\vdash \forall x\ y \bullet t\ x\ y = t\ y\ x$.
3. A theorem of the form $\vdash \forall x\ y\ z \bullet (x\ op\ y)\ op\ z = x\ op\ y\ op\ z$.
4. A conversion to be applied to each subterm of the form: $t_i\ op\ ...$ whenever such a subterm is created. The result of the conversion will not be further normalised.
5. A conversion to be applied to each atom as it is encountered. If the conversion produces a non-atomic term, this is normalised recursively.

The conversions supplied as parameters may signal that they do not wish to change a subterm, t say, either by failing or by returning $t = t$, the former approach is more efficient. The whole conversion fails with error number 81025 if there are no changes to be made to the term.

Errors

81021 ?0 does not have the form ⊢ t1 op t2 = t2 op t1
81022 ?0 does not have the form ⊢ (t1 op t2) op t3 = t1 op (t2 op t3)
81025 ?0 is already sorted
81029 Internal error: unexpected error in term normalisation package

SML

```
val term_order : TERM -> TERM -> int;
```

Description *term_order* gives an ordering relation on HOL terms. The ordering relation follows the same conventions as those used by the sorting function *sort*, namely, *term_order t1 t2* is negative if *t1* precedes *t2*, *0* if *t1* and *t2* are equivalent and positive if *t2* precedes *t1*. The ordering used is, with some exceptions, that all constants precede all variables which precede all abstractions which precede all applications. Lexicographic ordering on the immediate constituents gives the ordering within each of these four classes (using alphabetic ordering of strings, *type_order* or *term_order* recursively to order the constituents as appropriate). The exceptions are *(i)* that any term of the form $\neg t$ comes immediately after t, *(ii)* that the numeric literals *0*, *1*, ... are taken in numeric rather than alphabetic order and come before all other terms, and *(iii)* that terms of the form $i * x$ where i is a numeric literal are ordered so that the terms x, $0{*}x$, $1{*}x$, $2{*}x$, ... are consecutive.

See Also *gen_term_order1* which is the recommended way of constructing new term orderings.

SML

```
val type_order : TYPE -> TYPE -> int;
```

Description *type_order* gives a useful ordering relation HOL types. The ordering relation follows the same conventions as those used by the sorting function *sort*, namely, *type_order t1 t2* is negative if *t1* precedes *t2*, *0* if *t1* and *t2* are equivalent and positive if *t2* precedes *t1*. The ordering used is essentially that type variables are ordered by the alphabetic ordering of their names and precede all compound types which are ordered by the lexicographic ordering on their immediate constituents (using the alphabetic ordering for the type constructor names and the type ordering recursively for its operands).

8.6 First Order Resolution

SML

signature **Resolution** = *sig*

Description This is the signature of a structure providing Resolution facilities to ICL HOL.

SML

(∗ *resolution_diagnostics* − *boolean flag declared by new_flag* ∗)

Description This is by default false, but if set true then the resolution mechanism will report the generation of new, unsubsumed theorems, and whether these subsume pre-existing theorems.

Uses Provide the designer of the resolution functions access to detailed diagnostics. Not intended for use by others. May be withdrawn.

SML

```
type BASIC_RES_TYPE
        (* TERM * bool * (TERM * (TERM -> THM -> THM))list
           * TYPE list * THM * TERM list * TYPE list * int
           * FRAG_PRIORITY
        *);
type RES_DB_TYPE (* = BASIC_RES_TYPE list * BASIC_RES_TYPE list *
        BASIC_RES_TYPE list * THM list *);
```

Description These are type abbreviation for the basic resolution tool based on *prim_res_rule*. The arguments to *BASIC_RES_TYPE* are:

1. The term is a subterm of the theorem argument(5), reached through outer universal quantifications and all propositional connectives.
2. The bool is false if and only if the subterm occurs "negatively" in the conclusion of the theorem.
3. This list states how to specialise the given term to some other value in a theorem already specialised by the preceding entries in the list, and appropriately type instantiated.
4. The type list is the instantiable type variables of the subterm.
5. The theorem is the source of the fragment.
6. The term list is the term variables that may not be used in unifying the fragment
7. The next type list is the type variables that may not be used in unifying the fragment
8. The integer indicates the "generation", i.e. the number of resolutions involved in creating the fragment (initial theorems are at 0).
9. This argument indicates the priority given to taking this fragment from the *toprocess* list to use next.

The arguments to *RES_DB_TYPE*:

1. Items yet to be checked against (*against*).
2. Items checked against, but to be rechecked against new items to check with (*done*).
3. Items to check with (*toprocess*).
4. Theorems used to derive current items (*dbdata*).

SML

```
val BASIC_RESOLUTION_T : int -> THM list -> (THM -> TACTIC) ->
        (THM -> TACTIC) -> TACTIC;
```

Description *BASIC_RESOLUTION_T limit thms thmtac1 thmtac2* (*seqasms*, *conc*) will first apply *thmtac1* to the negated goal, probably adding it into the assumption list in some manner. The assumptions derived from this will become the set of support, the pre-existing assumptions and the input *thms* will be the rest of the theorems. These theorems will be resolved against each other until only theorems with default generation past *limit* can be derived, or until ...$\vdash$ *F* is derived, or until no further resolution can be done. An assumption's or input theorem's generation is *0*, and a theorem that is the result of resolution has a default generation of *1* plus the sum of the generations of the resolved theorems. Its actual generation will be the minimum of its default generation, and the generations of any previous theorems it subsumes. In any resolution attempt at least one of the two theorems will be assumed fragments from the stripped goal, or be derived from an earlier resolution in the evaluation. Duplicates and pure specialisations will be discarded.

Resolution will be attempted on subterms reached through outer universal quantification, and propositional connectives, by specialising the outer quantifications, and by type instantiation on the input theorems *thms* where necessary and possible.

The resulting list of theorems will have all the *thms* removed, all the theorems derived from stripping and negating the goal, and all the old assumptions removed. *MAP_EVERY thmtac* is then applied to the new theorems, and then to the goal. As a special case, ...$\vdash$ *F* is checked for, before any further processing. If present it will be used to prove the goal.

Uses On its own, or in combination with some canonicalisation of the input theorems.

Errors

67003 The limit, ?0, must be a positive integer
67004 No resolution occurred

SML

```
val BASIC_RESOLUTION_T1 : int -> THM list -> (THM -> TACTIC) ->
        TACTIC;
```

Description *BASIC_RESOLUTION_T1 limit thms thmtac* (*seqasms*, *conc*) will take the theorems gained by *asm_rule*'ing the assumptions and *thms* as inputs. These theorems will be resolved against each other until only theorems with default generation past *limit* can be derived, or until ...$\vdash$ *F* is derived, or until no further resolution can be done. An assumption's or input theorem's generation is *0*, and a theorem that is the result of resolution has a default generation of *1* plus the sum of the generations of the resolved theorems. Its actual generation will be the minimum of its default generation, and the generations of any previous theorems it subsumes. In any resolution attempt at least one of the two theorems will be from the original goals assumptions, or be derived from an earlier resolution in the evaluation. Duplicates and pure specialisations will be discarded.

Resolution will be attempted on subterms reached through outer universal quantification, and propositional connectives, by specialising the outer quantifications, and by type instantiation on the input theorems *thms* where necessary and possible.

The resulting list of theorems will have all the *thms* removed, and all the old assumptions removed. *MAP_EVERY thmtac* is then applied to the new theorems, and then to the goal. As a special case, ...$\vdash$ *F* is checked for, before any further processing. If present it will be used to prove the goal.

Uses On its own, or in combination with some canonicalisation of the input theorems.

Errors

67003 The limit, ?0, must be a positive integer
67004 No resolution occurred

SML

```
val basic_resolve_rule: TERM -> THM -> THM -> THM;
```

Description *basic_resolve_rule subterm pos neg* attempts to resolve two theorems that have a common subterm, *subterm*, occurring "positively" in *pos* and "negatively" in *neg*.

Rule

$$\frac{\Gamma \vdash P\ [subterm] \qquad \Delta \vdash N\ [subterm]}{simplify\ (\Gamma,\ \Delta \vdash P[F] \lor N[T])} \quad basic\_resolve\_rule\ subterm$$

Where *simplify* carries out the simplifications in the predicate calculus where an argument is the constant ⌜*T*⌝ or ⌜*F*⌝, plus a few others.

Errors

3031 ?0 is not of type ⌜:BOOL⌝
67009 ?0 is not a subterm of ?1

SML

```
val basic_res_extract : RES_DB_TYPE -> THM list;
```

Description This is the extraction function for the basic resolution tool based on *prim_res_rule*. It does no more than return the fourth item of the *RES_DB_TYPE* tuple.

SML
```
val basic_res_next_to_process : BASIC_RES_TYPE list ->
        BASIC_RES_TYPE list;
```

Description This takes as the next fragment to process the first fragment which comes from a theorem that subsumed some pre-existing one, and failing that the next one on the list of fragments.

SML
```
val basic_res_post :
        (THM -> THM -> int) ->
        (THM list * int) * RES_DB_TYPE ->
        (RES_DB_TYPE * bool);
```

Description This is the post processor for the basic resolution tool based on *prim_res_rule*. The results will be split into their respective conjuncts (if any). Then *basic_res_post subsum* ((*res*, *gen*), *data*) will test each member of *res*, checking for the conclusion *T* or *F*, and then against each member of the theorem list of *data*. In checking one theorem against another it will use *subsum* - discarding the new theorem if the result is *1*, and discarding (with tidying up of *data*) the original if the result is *2*, or keeping both (except for discards from further tests) if the result is *0*, or any other value bar *1* and *2*. *gen* is the default "generation" of the new theorems, except that the fragments for each new theorem will have the minimum generation number of this default generation, and the generation of any theorem in *data* it subsumes.

SML
```
val basic_res_pre : THM list -> THM list -> RES_DB_TYPE;
```

Description This is the preprocessor for the basic resolution tool based on *prim_res_rule*. The first argument is the set of support theorems, the second argument is the rest of the input theorems. Each theorem will be fragmented, and each fragment added to the appropriate list (i.e. to the third list of the result if in the set of support, and the first list if otherwise). The final theorem list part of the result, *dbdata*, is just the appending of the first list of theorems to the second.

SML
```
val basic_res_resolver : Unification.SUBS -> int ->
        BASIC_RES_TYPE -> BASIC_RES_TYPE -> THM list * int;
```

Description This is the resolver for the basic resolution tool based on *prim_res_rule*. Resolution seeks to find sufficient term specialisation and type instantiation on both terms to make one of the two term fragments the negation of the other, using *term_unify*. The resolution will not be attempted if the result would involve more resolutions than the "generations" limit. If this can be done then the two original theorems are specialised and instantiated in the same manner and the term fragment cancelled by *basic_resolve_rule*, and the result returned as a singleton list, paired with the default generation of the result. Prior to being returned, any allowed universal quantification will be added back in. In the basic resolution tool the generality of a list of theorems is unnecessary.

The *SUBS* argument is a "scratchpad" for the type unifier. The function keeps track of the number of resolutions used to create the result.

Errors
```
67001 Neither argument is in the set of support
67002 Cannot resolve the two arguments
67008 term_unify succeeded on ?0 and ?1 but failed to resolve ?2 and ?3
```

Message is a variant on 67002, included for diagnostic purposes. It will be removed in a more stable product.

SML
```
val basic_res_rule : int -> THM list -> THM list ->
        THM list;
```

Description *basic_res_rule limit sos rest* will resolve the theorems in the set of support and the rest against each other until only theorems with default generation past *limit* can be derived, or until ...⊢ F is derived, or until no further resolution can be done. A input theorem's generation is *0*, and a theorem that is the result of resolution has a default generation of *1* plus the sum of the generations of the resolved theorems. Its actual generation will be the minimum of its default generation, and the generations of any previous theorems it subsumes. In any resolution attempt at least one of the two theorems will belong to the set of support, or be derived from an earlier resolution in the evaluation. Resolution will be attempted on subterms reached through outer universal quantification, and propositional connectives, by specialising the outer quantifications, and by type instantiation where necessary and allowed. Duplicates and pure specialisations in the resulting list will be discarded.

If any of the input theorems have ⌜F⌝ as a conclusion then that theorem is returned as a singleton list.

Uses On its own, or in combination with some canonicalisation of the input theorems.

Errors
```
67003 The limit, ?0, must be a positive integer
67004 No resolution occurred
```

SML
```
val basic_res_subsumption : THM -> THM -> int;
```

Description This returns *1* if the conclusion of the first theorem equals the second's, or is a less general form than the second (i.e. could be produced only by specialising and type instantiating the second theorem). It returns *2* if the second theorem's conclusion is a less general form than the first, and otherwise returns *0*.

SML
```
val basic_res_tac1 : int -> THM list -> TACTIC;
```

Description *basic_res_tac1 limit thms* (*seqasms*, *conc*) will take the theorems gained by *asm_rule*'ing the assumptions and *thms* as inputs. These theorems will be resolved against each other until only theorems with default generation past *limit* can be derived, or until ...⊢ *F* is derived, or until no further resolution can be done. An assumption's or input theorem's generation is *0*, and a theorem that is the result of resolution has a default generation of *1* plus the sum of the generations of the resolved theorems. Its actual generation will be the minimum of its default generation, and the generations of any previous theorems it subsumes. In any resolution attempt at least one of the two theorems will be from the original goals assumptions, or be derived from an earlier resolution in the evaluation. Duplicates and pure specialisations will be discarded.

Resolution will be attempted on subterms reached through outer universal quantification, and propositional connectives, by specialising the outer quantifications, and by type instantiation on the input theorems *thms* where necessary and possible.

The resulting list of theorems will have all the *thms* removed, and all the old assumptions removed. *MAP_EVERY strip_asm_tac* is then applied to the new theorems, and then to the goal. As a special case, ...⊢ *F* is checked for, before any further processing. If present it will be used to prove the goal.

Uses On its own, or in combination with some canonicalisation of the input theorems.

Errors
67003 The limit, ?0, must be a positive integer
67004 No resolution occurred

SML
```
val basic_res_tac2 : int -> THM list -> TACTIC;
```

Description *basic_res_tac2 limit thms* (*seqasms*, *conc*) will first strip the negated goal into the assumption list. This uses *strip_tac*, except that the negation is pushed through all the outer universals. The assumptions derived from this will become the set of support, the pre-existing assumptions and the input *thms* will be the rest of the theorems. These theorems will be resolved against each other until only theorems with default generation past *limit* can be derived, or until ...⊢ *F* is derived, or until no further resolution can be done. An assumption's or input theorem's generation is *0*, and a theorem that is the result of resolution has a default generation of *1* plus the sum of the generations of the resolved theorems. Its actual generation will be the minimum of its default generation, and the generations of any previous theorems it subsumes. In any resolution attempt at least one of the two theorems will be assumed fragments from the stripped goal, or be derived from an earlier resolution in upon those fragments. Duplicates and pure specialisations will be discarded.

Resolution will be attempted on subterms reached through outer universal quantification, and propositional connectives, by specialising the outer quantifications, and by type instantiation on the input theorems *thms* where necessary and possible.

The tactic will fail unless the resulting list of theorems contains ...⊢ *F*. If present it will be used to prove the goal.

Errors
67003 The limit, ?0, must be a positive integer
67004 No resolution occurred
67014 Failed to prove goal

SML
```
val basic_res_tac3 : int -> THM list -> TACTIC;
```

Description *basic_res_tac3 limit thms* (*seqasms*, *conc*) will take the theorems gained by *asm_rule*'ing the assumptions and *thms* as inputs. These theorems will be resolved against each other until only theorems with default generation past *limit* can be derived, or until ...⊢ *F* is derived, or until no further resolution can be done. An assumption's or input theorem's generation is *0*, and a theorem that is the result of resolution has a default generation of *1* plus the sum of the generations of the resolved theorems. Its actual generation will be the minimum of its default generation, and the generations of any previous theorems it subsumes. In any resolution attempt at least one of the two theorems will be from the original goals assumptions, or be derived from an earlier resolution in the evaluation. Duplicates and pure specialisations will be discarded.

Resolution will be attempted on subterms reached through outer universal quantification, and propositional connectives, by specialising the outer quantifications, and by type instantiation on the input theorems *thms* where necessary and possible.

The tactic will fail unless the resulting list of theorems contains ...⊢ *F*. If present it will be used to prove the goal.

Errors
```
67003 The limit, ?0, must be a positive integer
67004 No resolution occurred
67014 Failed to prove goal
```

SML
```
val basic_res_tac4 : int -> int list -> int list ->
        THM list -> THM list -> TACTIC;
```

Description *basic_res_tac4 limit sos rest sos_thms rest_thms* (*seqasms*, *conc*) will take the theorems gained by *asm_rule*'ing the numbered assumptions and *thms* as inputs. The "set of support" theorems with be those assumptions noted in the *sos* and those theorems in *sos_thms*, and "the rest" will be those assumptions noted in the *rest*, as well as *rest_thms*. These theorems will be resolved against each other until only theorems with default generation past *limit* can be derived, or until ...⊢ *F* is derived, or until no further resolution can be done. An assumption's or input theorem's generation is *0*, and a theorem that is the result of resolution has a default generation of *1* plus the sum of the generations of the resolved theorems. Its actual generation will be the minimum of its default generation, and the generations of any previous theorems it subsumes. In any resolution attempt at least one of the two theorems will be from the set of support, or be derived from an earlier resolution in the evaluation. Duplicates and pure specialisations will be discarded.

Resolution will be attempted on subterms reached through outer universal quantification, and propositional connectives, by specialising the outer quantifications, and by type instantiation on the input theorems *thms* where necessary and possible.

The resulting list of theorems will have all the *thms* removed, and all the old assumptions removed. *MAP_EVERY strip_asm_tac* is then applied to the new theorems, and then to the goal. As a special case, ...⊢ *F* is checked for, before any further processing. If present it will be used to prove the goal.

Uses On its own, or in combination with some canonicalisation of the input theorems.

Errors
```
67003 The limit, ?0, must be a positive integer
67004 No resolution occurred
9303  the index ?0 is out of range
```

SML

|*val* **basic_res_tac** : *int* −> *THM list* −> *TACTIC*;

Description *basic_res_tac limit thms* (*seqasms*, *conc*) will first strip the negated goal into the assumption list. This uses *strip_tac*, except that the negation is pushed through all the outer universals. The assumptions derived from this will become the set of support, the pre-existing assumptions and the input *thms* will be the rest of the theorems. These theorems will be resolved against each other until only theorems with default generation past *limit* can be derived, or until ...⊢ *F* is derived, or until no further resolution can be done. An assumption's or input theorem's generation is *0*, and a theorem that is the result of resolution has a default generation of *1* plus the sum of the generations of the resolved theorems. Its actual generation will be the minimum of its default generation, and the generations of any previous theorems it subsumes. In any resolution attempt at least one of the two theorems will be assumed fragments from the stripped goal, or be derived from an earlier resolution in upon those fragments. Duplicates and pure specialisations will be discarded.

Resolution will be attempted on subterms reached through outer universal quantification, and propositional connectives, by specialising the outer quantifications, and by type instantiation on the input theorems *thms* where necessary and possible.

The resulting list of theorems will have all the *thms* removed, all the theorems derived from stripping and negating the goal removed, and all the old assumptions removed. *MAP_EVERY strip_asm_tac* is then applied to the new theorems, and then to the goal. As a special case, ...⊢ *F* is checked for, before any further processing. If present it will be used to prove the goal.

Uses On its own, or in combination with some canonicalisation of the input theorems.

Errors

|*67003 The limit, ?0, must be a positive integer*
|*67004 No resolution occurred*

SML

```
val prim_res_rule :
        (THM list -> THM list -> ('a list * 'a list * 'a list * 'b)) -> (* preprocessor *)
        ('a -> 'a -> 'c) -> (* the resolver function *)
        (('c * ('a list * 'a list * 'a list * 'b)) ->
                (('a list * 'a list * 'a list * 'b) * bool)) -> (* postprocessor *)
        ('a list -> 'a list) -> (* next item to process *)
        ('a list * 'a list * 'a list * 'b -> THM list) -> (* extract results *)
        THM list -> (* input set of support theorems *)
        THM list -> (* input other theorems *)
        THM list;   (* final outcome *)
```

Description *prim_res_rule prep reso postp next extract limit sos rest* works as follows:

- If any of the input theorems have ⌜F⌝ as a conclusion then that theorem is returned as a singleton list.
- Evaluate *prep sos rest*, and set (*against*, *tried*, *toprocess*, *dbdata*) to this.
- Attempt resolutions, choosing the head of *toprocess* against the head of *against*. Commonly, the head of *toprocess* should be the first fragment from the set of support, *against* is all the non-set of support fragments, plus the head of *toprocess*, and *tried* is empty.
- The resolver will usually return a list of theorems, and perhaps some further data. When a resolution attempt returns a list of theorems, *res*, (resolution failures should not occur, just []), evaluate *postp* (*res*, (*against*, *tried*, *toprocess*, *dbdata*)) to extract a new (*against*, *tried*, *toprocess*, *dbdata*), and *halt*. It is up to the postprocessor to move the head of *against* either to *tried* or just thrown away.
- If *halt* is true (e.g. have proved ...$\vdash F$), or the *toprocess* list is empty then return as a result of the call *extract* (*against*, *tried*, *toprocess*, *dbdata*).
- If *halt* is false, then continue with the new data. If *against* is [] then the head of *toprocess* is dropped, and the new list of things to process generated by *next* (*tl toprocess*), the new head of this cons'd to *done* and *against* is set to *done* reversed, and then *done* set to [].

Errors

67004	*No resolution occurred*
67010	*Postprocessor corrupted processing*

SML

```
val term_unify : Unification.SUBS -> (TYPE list) -> (TERM list) ->
        (TERM * TERM list * TYPE list) *
        (TERM * TERM list * TYPE list) ->
        ((TYPE * TYPE) list * (TERM * TERM) list) *
        ((TYPE * TYPE) list * (TERM * TERM) list);
```

Description This is a method of unifying two subterms in the context of limitations on both type instantiation and term specialisation. The *SUBS* argument is a "scratchpad" for the type unification function, based on *Unification.unify*. The initial type list is a list of type variables to avoid in generating new names, and the initial term list a list of term variables to likewise avoid. The other two input arguments are each a tuple of: a term to unify, a list of variables in the term that may be specialised, and a list of types for which instantiation is allowed. If the two terms can be unified then the function returns two tuples, referring to each of the two input tuples. Each tuple is a list of type instantiations and a list of term specialisations, which pair the original before type instantiation, and the result, type instantiated.

Errors

3007 ?0 is not a term variable
3019 ?0 is not a type variable
67005 Cannot unify ?0 and ?1
67006 Cannot unify ?0 and ?1 as cannot specialise ?2
67012 Cannot unify ?0 and ?1 as would cause a loop

As as errors of *Unification.unify*.

8.7 Quantifier Elimination Toolkit

SML

val **dest_quant** : *TERM -> TERM * TERM * TERM*;
val **is_quant** : *TERM -> bool*;
val **mk_quant** : *TERM * TERM * TERM -> TERM*;

Description These functions are the destructor, discriminator and constructor functions for quantified terms, i.e., boolean terms formed by applying a constant to a (possibly paired) λ-abstraction.

Errors

119001 ?0 is not a quantified term
119003 ?0 is not a valid varstruct
119004 ?0 is not of type ⌜:BOOL⌝

SML

val **FAIL_C** : *CONV -> CONV*;

Description Conversional that always fails: *ID_C conv* is the same as *fail_conv*. This is for use with other conversionals which take a conversional as an input.

SML

val **find_in_∧_conv** : *(TERM -> bool) -> CONV*;

Description Given a function specifying a property of terms and an iterated conjunction, $a \wedge b \wedge ...$, this conversion rearranges it in the form: $c \wedge a \wedge b \wedge ...$ where c is the first conjunct satisfying the property. It fails if there is nothing to do or if no conjunct satisfies the property.

Errors

119008 ?0 cannot be rearranged or is already in the required form

SML

val **FIRST_THEN_C** : *(CONV * (CONV -> CONV)) list -> CONV*;

Description *FIRST_THEN_C* takes a list of conversion-conversional pairs and gives which acts as follows: the conversions in the list are tried one after the other, and the process only fails if none of the conversions is applicable; when a conversion succeeds, *FIRST_THEN_C* is applied recursively to the remainder of the list as if by *X_C FIRST_THEN_C*, where *X_C* is the conversional associated with the successful conversion. The conversionals are intended to allow the process to be diverted into subterms of the result of a transformation. For example, one might transform a unique existential quantification into an existential whose matrix contains a universal and then divert attention to the universal.

Errors

119010 None of the conversions was applicable to ?0

SML

val **ID_C** : *CONV -> CONV*;

Description The identity conversional: *ID_C conv* is the same as *conv*. This is for use with other conversionals such as *FIRST_THEN_C*, q.v., which take a conversional as an input.

SML

val **INNERMOST_QUANT_C** : *CONV -> CONV*;

Description *INNERMOST_QUANT_C* is a conversional that maps a conversion over the innermost quantified subterms of the first-order structure of a term. It fails if the first-order structure of the term is quantifier-free.

Errors

119011 No quantified subterms were found

SML
```
val nnf_conv : CONV;
```

Description This conversion puts a propositional formula into negation normal form, i.e., a normal form in which ¬ is pushed through the other propositional connectives using the following rules:

$$\begin{array}{ll}
\neg(p1 \land p2) \Leftrightarrow \neg p1 \lor \neg p2 & \neg(p1 \lor p2) \Leftrightarrow \neg p1 \land \neg p2 \\
\neg(p1 \Rightarrow p2) \Leftrightarrow p1 \land \neg p2 & \neg(p1 \Leftrightarrow p2) \Leftrightarrow p1 \land \neg p2 \lor \neg p1 \land p2 \\
(p1 \Leftrightarrow p2) \Leftrightarrow p1 \land p2 \lor \neg p1 \land \neg p2 & \neg(\neg p) \Leftrightarrow p
\end{array}$$

Propositional simplification using *prop_simp_conv* is attempted before and after the above rules.

Errors
119007 ?0 is not of type ⌜:BOOL⌝ or is already in negation normal form

SML
```
val prenex_clauses : THM;
```

Description Higher-order rewriting with this theorem carries out conversion to prenex normal form.

SML
```
val PROP_ATOM_C : CONV -> CONV;
val PROP_LIT_C : CONV -> CONV;
```

Description *PROP_ATOM_C* is a conversional that maps its argument over the outermost propositional atoms in a term. *PROP_LIT_C* maps its argument over the outermost propositional literals in a term. Here a propositional atom is any term other than one formed using one of the propositional connectives ∧, ∨, ⇒, ⇔ or ¬. A propositional literal is either a propositional atom or the negation of a propositional atom.

SML
```
val prop_simp_conv : CONV;
```

Description This conversion simplifies propositional formula using the following rules:

$$\begin{array}{llll}
\neg(\neg p) \Leftrightarrow p & \neg F \Leftrightarrow T & \neg T \Leftrightarrow F & \\
F \lor p \Leftrightarrow p & p \lor F \Leftrightarrow p & T \lor p \Leftrightarrow T & p \lor T \Leftrightarrow T \\
p \land F \Leftrightarrow F & F \land p \Leftrightarrow F & T \land p \Leftrightarrow p & p \land T \Leftrightarrow p \\
p \Rightarrow T \Leftrightarrow T & T \Rightarrow p \Leftrightarrow p & F \Rightarrow p \Leftrightarrow T & p \Rightarrow F \Leftrightarrow \neg p \\
\multicolumn{3}{l}{(p \Leftrightarrow T) \Leftrightarrow p\ (T \Leftrightarrow p) \Leftrightarrow p\ (F \Leftrightarrow p) \Leftrightarrow \neg p} & (p \Leftrightarrow F) \Leftrightarrow \neg p \\
(p \lor p) \Leftrightarrow p & p \land p \Leftrightarrow p & \multicolumn{2}{l}{(p \Rightarrow p) \Leftrightarrow T\ (p \Leftrightarrow p) \Leftrightarrow T} \\
(p \lor \neg p) \Leftrightarrow T & p \land \neg p \Leftrightarrow F & (p \Rightarrow \neg p) \Leftrightarrow \neg p & (p \Leftrightarrow \neg p) \Leftrightarrow F \\
(\neg p \lor p) \Leftrightarrow T & \neg p \land p \Leftrightarrow F & (\neg p \Rightarrow p) \Leftrightarrow p & (\neg p \Leftrightarrow p) \Leftrightarrow F
\end{array}$$

The conversion fails if no simplification is possible.

Errors
119012 No simplifications apply to ?0

SML
```
val QUANTS_C : CONV -> CONV;
```

Description *QUANTS_C* is a conversional that maps its argument conversion over a term formed by repeatedly applying quantifiers to a non-quantified term. The conversion is applied to each quantified term obtained by stripping off a leading part of the prefix, the innermost being processed first.

SML
```
val QUANT_C : CONV -> CONV;
```

Description $QUANT\_C$ is a conversional that applies its argument to the matrix of a quantified term.

SML
```
val QUANT_MAP_C : CONV -> CONV;
```

Description $QUANT\_MAP\_C$ is a conversional that maps a conversion over all the quantifiers in a term, innermost quantifiers first. If the conversion succeeds it is retried, so it should fail if it cannot make any progress.

SML
```
val simple_one_point_conv : CONV;
```

Description This conversion proves all theorems of the forms $\vdash (\exists x \bullet x = t \land p) \Leftrightarrow p[t/x]$ and $\vdash (\exists x \bullet x = t) \Leftrightarrow T$ where $p[t/x]$ denotes the result of substituting the term t for the variable x in p. In the second form, t may be the same as x.

Errors
```
119009 ?0 is not of the form ⌜∃x•x = t ∧ p⌝ or ⌜∃x•x = t⌝
```

SML
```
val simple_∀_¬_∃_¬_conv : CONV;
```

Description This conversion proves theorems of the form $\vdash (\forall x \bullet t) \Leftrightarrow (\neg(\exists x \bullet \neg t))$

Errors
```
119005 ?0 is not of the form ⌜∀x•t⌝ where ⌜x⌝ is a variable
```

SML
```
val simple_∃_const_elim_conv : CONV;
```

Description This conversion proves all theorems of the form: $\vdash (\exists x \bullet p) \Leftrightarrow p$ where the variable x does not appear free in p.

Errors
```
119006 ?0 is not of the form ⌜∃x•p⌝ where ⌜x⌝ is a variable
          that does not occur free in ⌜p⌝
```

SML
```
val split_∧_conv : (TERM -> bool) -> CONV;
```

Description Given a function specifying a property of terms and an iterated conjunction, $a \land b \land ...$, this conversion rearranges the conjunction up into the form: $(d \land c \land ...) \land (e \land f \land ...)$, where $c, d, \ldots$ satisfy the property and $e, f, \ldots$ do not. It fails there is nothing to do, i.e., if all the conjuncts satisfy the property, if none of the conjuncts satisfy the property or if the first conjunct satisfies the property and the others do not.

8.8 Proof Contexts

SML
signature **ProofContext** = *sig*

Description This provides the basic tools for handling equational and proof contexts. To keep them short, the names in the structure are heavily abbreviated. The abbreviations used are:

pc(s)	proof context(s)
rw	rewriting
cs	constant specification
∃	existential theorem prover
pr	*prove_tac* and related tools
sg	goal stripping
st	theorem stripping
cd	clausal definition
vs	variable structure
ad	application data
net	discrimination net
eqn_cxt	equational context
nd	dictionary of discrimination nets (and sources)
canon(s)	theorem canonicalisation
mmp	matching mp rule
eqm	equation matcher

SML
(∗ *proof context key* **"initial"** ∗)

Description This is the initial proof context, formed with empty lists and other default values. It thus has no default rewriting or stripping theorems. The rewriting canonicalisation is the identity. The automated existence prover fails on any input. The matching modus ponens rule is *Nil*.

SML
type **EQN_CXT**;

Description This is the type of equational contexts. An equational context is a list of conversions, each paired with term index. It represents a statement of how to rewrite a term to result in an equational theorem, guided by the outermost form of the term to be rewritten, which is matched against the term index of each conversion. It is used to create a single conversion via *eqn_cxt_conv* (q.v.).

A theorem may be converted into a member of an equational context by *thm_eqn_cxt*. A preexisting conversion may be converted by determining the term index that matches at least all terms that the conversion must work on (see *net_enter* for details), and pair it with the conversion.

type EQN_CXT = (*TERM* ∗ *CONV*) *list*;

Note that equational contexts can be merged by appending. An equational context may be transformed into a conversion discrimination net by *make_net* or *list_net_enter*(q.v.).

SML
```
val asm_prove_tac : THM list -> TACTIC;
```

Description This tactic is an automatic proof procedure appropriate to the current proof context.

At the point of applying this tactic to its theorems it will access the current setting of proof context field *pr_tac*, apply it to the theorem list immediately, and then to the goal when available (i.e. the result is partially evaluated with only the list of theorems).

Tactic

$$\frac{\{\ \Gamma\ \}\ t}{current\_ad\_pr\_tac\ ()\ thms\ (\{\ \Gamma\ \},\ t)} \quad asm\_prove\_tac\ thms$$

See Also *PC_T1* to defer accessing the proof context until application to the goal; *prove_tac* for the form that does not react to the presence of assumptions.

Errors

51021 The current proof context was created in theory ?0 at a point now either not in scope, deleted or modified

and as the proof context setting.

SML
```
val asm_prove_∃_tac : TACTIC;
```

Description This tactic is an automatic proof procedure for existential proofs, appropriate to the current proof context.

At the point of applying this tactic to a goal it will access the current setting of proof context field *prove_∃*, apply it to the goal using *conv_tac*.

Tactic

$$\frac{\{\ \Gamma\ \}\ t}{conv\_tac\ (current\_ad\_cs\_\exists\_conv\ ())\ (\{\ \Gamma\ \},\ t)} \quad asm\_prove\_\exists\_tac\ thms$$

See Also *prove_∃_tac* that does not react to any assumptions that are present.

Errors

51021 The current proof context was created in theory ?0 at a point now either not in scope, deleted or modified

and as the proof context setting.

SML
```
val commit_pc : string -> unit;
```

Description This commits a record of the proof context database, preventing further change, and allowing it to be used in the creation of further records. The context must be loadable at the point of committing (i.e. was created at a point now in scope), and after committal the proof context can only be loaded at a point when the point of committal is in scope, rather than the point of its initial creation (i.e. doing *new_pc*).

Errors

51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified
51016 Proof context ?0 has been committed

SML
```
val current_ad_mmp_rule : unit -> (THM -> THM -> THM) OPT;
```

Description This function returns the application data of the current proof context for the matching modus ponens rule as used by tools such as *forward_chain_rule*.

See Also *set_mmp_rule* for user data.

Errors
```
51021 The current proof context was created in theory ?0 at a
      point now either not in scope, deleted or modified
```

SML
```
val current_ad_pr_conv : unit -> (THM list -> CONV) OPT;
```

Description These functions returns the application data of the current proof context to the proof contexts for *prove_conv*.

See Also *set_pr_conv* for user data.

Errors
```
51021 The current proof context was created in theory ?0 at a
      point now either not in scope, deleted or modified
```

SML
```
val current_ad_pr_tac : unit -> (THM list -> TACTIC) OPT;
```

Description This function returns the application data of the current proof context for *prove_tac*.

See Also *set_pr_tac* for user data.

Errors
```
51021 The current proof context was created in theory ?0 at a
      point now either not in scope, deleted or modified
```

SML
```
val current_ad_rw_eqm_rule : unit -> (THM -> TERM * CONV) OPT;
```

Description This function returns the application data of the current proof context for the equation matcher as used by the rewriting tools.

See Also *set_rw_eqm_rule* for user data.

Errors
```
51021 The current proof context was created in theory ?0 at a
      point now either not in scope, deleted or modified
```

SML
```
val delete_pc_fields : string list -> string -> unit;
```

Description *delete_pc_fields fields key* empties (sets to the value of proof context "initial") the named fields, *fields* of the proof context with key *key*. If any field is divided into subfields, this deletion includes deleting the subfields of the field gained from merging in other proof contexts, as well as the proof context's "own" subfield.

Valid field names are:

```
"rw_eqn_cxt","rw_canons","st_eqn_cxt","sc_eqn_cxt",
"cs_∃_convs","∃_cd_thms","∃_vs_thms","pr_tac","pr_conv",
"nd_entries", "mmp_rule"
```

Errors
```
51010  There is no proof context with key ?0
51014  Proof context ?0 was created in theory ?1 at a
       point now either not in scope, deleted or modified
51016  Proof context ?0 has been committed
51019  There is no field called ?0
```

SML
```
val delete_pc : string -> unit;
```

Description This deletes a record from the proof context database. The record with key "initial" may not be deleted.

Errors
```
51010  There is no proof context with key ?0
51012  Initial proof context may not be deleted
```

SML
```
val eqn_cxt_conv : EQN_CXT -> CONV;
```

Description This function creates a single conversion from an equational context. This is done via *make_net* and *net_lookup*(q.v). There is a *CHANGED_C* wrapped around each conversion in the equational context.

Errors
```
51005  Equational context gave no conversions that succeeded for ?0
```

SML
```
val EXTEND_PC_C1 : string -> ('a -> CONV) -> 'a -> CONV;
val EXTEND_PCS_C1 : string list -> ('a -> CONV) -> 'a -> CONV;
```

Description *EXTEND_PC_C context conv arg* will apply conversion *conv arg* in the proof context obtained by merging the proof context with key *context* into the current proof context. The named context is used as it is at the point of applying the rule to the argument. The *pr_tac*, *pr_conv* and *mpp_rule* fields are taken from the named proof context. This is done via pushing and popping on the proof context stack.

EXTEND_PCS_C1 takes a list of proof contexts instead, merged as if by, e.g. *push_extend_pcs*.

See Also *PC_C*

Errors
```
51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a
      point now either not in scope, deleted or modified
```

and as the errors of the conversion. The previous proof context is restored, even if the conversion fails.

SML
```
val EXTEND_PC_C : string -> CONV -> CONV;
val EXTEND_PCS_C : string list -> CONV -> CONV;
```

Description *EXTEND_PC_C context conv* will apply conversion *conv* to a term in the proof context obtained by merging the proof context with key *context* into the current proof context. The named context is used as it is at the point of applying the conversion to a term. The *pr_tac*, *pr_conv* and *mpp_rule* fields are taken from the named proof context. This is done via pushing and popping on the proof context stack.

EXTEND_PCS_C takes a list of proof contexts instead, merged as if by, e.g. *push_extend_pcs*

Note that when using this functions that the standard rewriting functions (obvious candidates for this function) access the current proof context at the point of being given their theorem list argument: see *EXTEND_PC_C1* for a method of avoiding this.

Errors
```
51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a
      point now either not in scope, deleted or modified
```

and as the errors of the conversion. The previous proof context is restored, even if the conversion fails.

SML
```
val extend_pc_rule1 : string -> ('a -> 'b -> THM) -> 'a -> 'b -> THM;
val extend_pcs_rule1 : string list -> ('a -> 'b -> THM) -> 'a -> 'b -> THM;
```

Description *extend_pc_rule1 context rule arg1 arg2* will apply rule *rule arg1* to *arg2* in the proof context obtained by merging the proof context with key *context* into the current proof context. The named context is used as it is at the point of applying the rule to the argument. The *pr_tac*, *pr_conv* and *mpp_rule* fields are taken from the named proof context. This is done via pushing and popping on the proof context stack.

extend_pcs_rule1 takes a list of proof contexts instead, merged as if by, e.g. *push_extend_pcs*.

See Also *pc_rule*

Errors

51010	*There is no proof context with key ?0*
51014	*Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified*

and as the errors of the rule. The previous proof context is restored, even if the rule fails.

SML
```
val extend_pc_rule : string -> ('a -> THM) -> ('a -> THM);
val extend_pcs_rule : string list -> ('a -> THM) -> ('a -> THM);
```

Description *extend pc_rule context rule* will apply rule *rule* to its argument in the proof context obtained by merging the proof context with key *context* into the current proof context. The named context is used as it is at the point of applying the rule to the argument. The *pr_tac*, *pr_conv* and *mpp_rule* fields are taken from the named proof context. This is done via pushing and popping on the proof context stack.

extend_pcs_rule takes a list of proof contexts instead, merged as if by, e.g. *extend_merge_pcs*

Note that when using this functions that the standard rewriting functions (obvious candidates for this function) access the current proof context at the point of being given their theorem list argument: see *extend_pc_rule1* for a method of avoiding this.

Errors

51010	*There is no proof context with key ?0*
51014	*Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified*

and as the errors of the rule. The previous proof context is restored, even if the rule fails.

SML

```
val EXTEND_PC_T1 : string -> ('a -> TACTIC) -> 'a -> TACTIC;
val EXTEND_PCS_T1 : string list -> ('a -> TACTIC) -> 'a -> TACTIC;
```

Description *EXTEND_PC_T1 context tac arg* will apply tactic *tac arg* to a goal, and evaluate the proof, in the proof context obtained by merging the proof context with key *context* into the current proof context. The named context is used as it is at the point of applying the rule to the argument. The *pr_tac*, *pr_conv* and *mpp_rule* fields are taken from the named proof context. This is done via pushing and popping on the proof context stack.

EXTEND_PCS_T1 takes a list of proof contexts instead, merged as if by, e.g. *push_extend_pcs*.

See Also *PC_T*

Errors

51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified

and as the errors of the tactic. The previous proof context is restored, even if the tactic application or proof fails.

SML

```
val EXTEND_PC_T : string -> TACTIC -> TACTIC;
val EXTEND_PCS_T : string list -> TACTIC -> TACTIC;
```

Description *EXTEND_PC_T context tac* will apply tactic *tac* to a goal, and evaluate its proof, in the proof context obtained by merging the proof context with key *context* into the current proof context. The named context is used as it is at the point of applying the tactic to a goal. The *pr_tac*, *pr_conv* and *mpp_rule* fields are taken from the named proof context. This is done via pushing and popping on the proof context stack.

EXTEND_PCS_T_T takes a list of proof contexts instead, merged as if by, e.g. *push_extend_pcs*

Note that when using this functions that the standard rewriting functions (obvious candidates for this function) access the current proof context at the point of being given their theorem list argument: see *EXTEND_PC_T1* for a method of avoiding this.

Errors

51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified

and as the errors of the tactic. The previous proof context is restored, even if the tactic application or proof fails.

SML
```
val force_delete_theory : string -> unit;
```

Description *force_delete_theory thy* attempts to delete theory *thy* and all its descendants. If *thy* is in scope, then the function will change the current theory to the first theory that it can in the list returned by *get_parents thy*; (there may be none, in which case the function fails). It will then determine whether *thy* and its descendants can all be deleted: in particular it checks that none of them are locked (see *lock_theory*) or are a read-only ancestor.

The function indicates:

- whether the current theory has been deleted, and if so states the new current theory,
- the list of theories that have been deleted (unless this is just the requested theory, and is also not the current theory).

Further, all proof contexts created in now deleted theories will also be deleted (but the current proof context will remain unchanged).

Errors
```
51002 Cannot open any of the parent theories, ?0, of the named theory, ?1
51003 Will not be able to delete theories ?0, so no deletions made
51004 Unexpectedly unable to delete any of ?0
51006 Cannot open the parent theory, ?0, of the named theory, ?1
51007 Will not be able to delete theory ?0, so no deletions made
51008 Named theory, ?0, has no parents
```

Error 51004 will be raised by *error* rather than *fail*, as it shouldn't happen.

SML
```
val get_current_pc : unit -> (string list * string);
```

Description Returns the key(s) of the entries from which the current proof context was copied, and the theory in which the single proof context was created. If the theory has since been placed out of scope, deleted or if the definition level becomes deleted, e.g. because an axiom or definition has been deleted, then this will output

```
(["context name"],"theory name (out of scope, deleted, or modified)")
```

Note that the key may no longer access a proof context in the database identical to the current proof context.

Merged proof contexts upon the stack (from *push_merge_pcs* and *set_merge_pcs*) will have the list of names of the constituent proof contexts, singleton contexts will have singleton lists.

See Also *get_stack_pcs*

SML
```
val get_pcs : unit -> (string * string) list;
```

Description This lists the names of the proof contexts held in the proof context database, and the theory that was current at their time of creation. If the theory has since been deleted or if the definition level becomes deleted, e.g. because an axiom or definition has been deleted, then this will output ("*context name*","*theory name* (*out of scope*, *deleted*, *or modified*)")

See Also *get_stack_pcs*, *get_current_pc*.

SML
```
val get_stack_pcs : unit -> (string list * string) list;
```

Description This lists the keys of the proof contexts held in the proof context stack, and the theory that was current at their time of creation. If a proof context is pushed onto the stack by, e.g. *push_pc*, the "keys" will be the singleton list of the name of the source proof context. If a proof context is pushed onto the stack by, e.g. *push_merge_pcs*, the "keys" will be the list of the names of the source proof contexts. If the theory has since been deleted or if the definition level becomes deleted, e.g. because an axiom or definition has been deleted, then this will output

```
(["context name"],"theory name (out of scope, deleted, or modified)")
```

The head of the list returned is the current proof context, as also displayed by *get_current_pc*.

SML
```
val merge_pcs : string list -> string -> unit;
```

Description *merge_pcs keys tokey* takes a list of committed proof contexts named by *keys*, and merges their fields into proof context *tokey*'s fields, discarding duplicates. For each field that has subfields the lists of subfields from each proof context are appended, discarding subfields with duplicate keys, and if a field is not divided into subfields, then the proof contexts fields are appended, discarding duplicates. The *pr_conv*, *pr_tac* and *mmp_rule* fields take the value of the last proof context in the list that has the field set.

Failure to extract any proof context for merging will result in the proof context *tokey* being unchanged.

See Also *merge_pc_fields*, *delete_pc_fields*

Errors
```
51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a
      point now either not in scope, deleted or modified
51016 Proof context ?0 has been committed
51017 Proof context ?0 has not been committed
```

SML
```
val merge_pc_fields : {context:string,fields:string list}list -> string -> unit;
```

Description *merge_pc_fields fields tokey* merges the fields noted for each committed proof context in *fields* into proof context *tokey*'s fields, discarding duplicates. Merging for each field that has subfields the lists of subfields is appending the proof contexts fields, discarding subfields with duplicate keys, and if a field is not divided into subfields, then the proof contexts fields are appended, discarding duplicates. Each of the *pr_conv*, *pr_tac* and *mmp_rule* fields take the value from the last proof context whose list of field names includes that field and which has the field set.

Failure to extract any proof context for merging will result in the proof context *tokey* being unchanged.

Valid field names are:

```
"rw_eqn_cxt","rw_canons","st_eqn_cxt","sc_eqn_cxt",
"cs_∃_convs","∃_cd_thms","∃_vs_thms","pr_tac","pr_conv",
"nd_entries", "mmp_rule"
```

See Also *delete_pc_fields* and *merge_pcs*, which used together in a particular order can give the same functionality as this function.

Errors
```
51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a
      point now either not in scope, deleted or modified
51016 Proof context ?0 has been committed
51017 Proof context ?0 has not been committed
51019 There is no field called ?0
```

SML
```
val new_pc : string -> unit;
```

Description *new_pc new* creates a new record in the proof context database, with key *new*. The fields of the proof context are set to default values. A note will be made of the current theory, and its current definition level at the time of creation, and an error will be raised if an attempt is made to push the new proof context (see *push_pc*) when that theory is not in scope, or when the definition level has been recorded as deleted. The definition level will be recorded as deleted if, e.g., some definition or axiom that was in scope in the original theory has since been deleted.

One responsibility of the creator of a proof context is to ensure that the theorems used within, or created by, the new context are also in scope: this is not automatically checked.

Errors
```
51011 There is already a proof context with key ?0
```

SML
```
val PC_C1 : string -> ('a -> CONV) -> 'a -> CONV;
val MERGE_PCS_C1 : string list -> ('a -> CONV) -> 'a -> CONV;
```

Description *PC_C context conv arg* will apply conversion *conv arg* in the proof context with key *context*, using the named context as it is at the point of applying the conversion to a term. This is done via pushing and popping on the proof context stack.

MERGE_PCS_C1 takes a list of proof contexts instead, merged as if by, e.g. *push_merge_pcs*.

See Also *PC_C*

Errors

51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified

and as the errors of the conversion. The previous proof context is restored, even if the conversion fails.

SML
```
val PC_C : string -> CONV -> CONV;
val MERGE_PCS_C : string list -> CONV -> CONV;
```

Description *PC_C context conv* will apply conversion *conv* to a term in the proof context with key *context*, using the named context as it is at the point of applying the conversion to a term. This is done via pushing and popping on the proof context stack.

MERGE_PCS_C takes a list of proof contexts instead, merged as if by, e.g. *push_merge_pcs*

Note that when using this functions that the standard rewriting conversions (obvious candidates for this function) access the current proof context at the point of being given their theorem list argument: see *PC_C1* for a method of avoiding this.

Errors

51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified

and as the errors of the conversion. The previous proof context is restored, even if the conversion fails.

SML
```
val pc_rule1 : string -> ('a -> 'b -> THM) -> 'a -> 'b -> THM;
val merge_pcs_rule1 : string list -> ('a -> 'b -> THM) -> 'a -> 'b -> THM;
```

Description *pc_rule context rule arg1 arg2* will apply rule *rule arg1* to *arg2* in the proof context with key *context*, using the named context as it is at the point of applying the rule to argument *arg2*. This is done via pushing and popping on the proof context stack.

merge_pcs_rule1 takes a list of proof contexts instead, merged as if by, e.g. *push_merge_pcs*.

See Also *pc_rule*

Errors

51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified

and as the errors of the rule. The previous proof context is restored, even if the rule fails.

SML

```
val pc_rule : string -> ('a -> THM) -> ('a -> THM);
val merge_pcs_rule : string list -> ('a -> THM) -> ('a -> THM);
```

Description *pc_rule context rule* will apply rule *rule* to its argument in the proof context with key *context*, using the named context as it is at the point of applying the rule to the argument. This is done via pushing and popping on the proof context stack.

merge_pcs_rule takes a list of proof contexts instead, merged as if by, e.g. *push_merge_pcs*

Note that when using this functions that the standard rewriting functions (obvious candidates for this function) access the current proof context at the point of being given their theorem list argument: see *pc_rule1* for a method of avoiding this.

Errors

51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified

and as the errors of the rule. The previous proof context is restored, even if the rule fails.

SML

```
val PC_T1 : string -> ('a -> TACTIC) -> 'a -> TACTIC;
val MERGE_PCS_T1 : string list -> ('a -> TACTIC) -> 'a -> TACTIC;
```

Description *PC_T1 context tac arg* will apply tactic *tac arg* to a goal, and evaluate the proof, in the proof context with key *context*, using at both times the named context as it is at the point of applying the tactic to a goal. This is done via pushing and popping on the proof context stack.

MERGE_PCS_T1 takes a list of proof contexts instead, merged as if by, e.g. *push_merge_pcs*.

See Also *PC_T*

Errors

51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified

and as the errors of the tactic. The previous proof context is restored, even if the tactic application or proof fails.

SML

```
val PC_T : string -> TACTIC -> TACTIC;
val MERGE_PCS_T : string list -> TACTIC -> TACTIC;
```

Description *PC_T context tac* will apply tactic *tac* to a goal, and evaluate its proof, in the proof context with key *context*, using at both times the named context as it is at the point of applying the tactic to a goal. This is done via pushing and popping on the proof context stack.

PCS_MERGE_T takes a list of proof contexts instead, merged as if by, e.g. *push_merge_pcs*

Note that when using this functions that the standard rewriting functions (obvious candidates for this function) access the current proof context at the point of being given their theorem list argument: see *PC_T1* for a method of avoiding this.

Errors

51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified

and as the errors of the tactic. The previous proof context is restored, even if the tactic application or proof fails.

SML
```
val pending_push_merge_pcs : string list -> unit -> unit;
val pending_push_extend_pcs : string list -> unit -> unit;
```

Description *pending_push_merge_pcs* takes a snapshot of the result of merging the named proof contexts, and returns a function that, when applied to () stacks the previous proof context, and and sets the current proof context of the system to this snapshot.

pending_push_extend_pcs takes a snapshot of the result of merging the named proof contexts with the current proof context and then behaves just like *pending_push_merge_pcs*.

Merged proof contexts upon the stack will have *current_ad_names* giving the list of names of the constituent proof contexts, singleton contexts will have singleton lists. The proof contexts used need not have been committed. The *pr_conv*, *pr_tac* and *mmp_rule* fields take the value of the last proof context in the list that has the field set.

The proof context must be in scope both at the time of the snapshot, and at the time of pushing on the stack.

This provides a method of being independent of changes to uncommitted proof contexts, or proof context deletions.

See Also *push_merge_pc*

Errors

51010	*There is no proof context with key ?0*
51014	*Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified*
51020	*Must be at least one key in list*

SML
```
val pending_push_pc : string -> unit -> unit;
val pending_push_extend_pc : string -> unit -> unit;
```

Description *pending_push_pc* takes a snapshot of the named proof context, and returns a function that, when applied to () : *unit* stacks the previous "current" proof context, and sets the current proof context of the system to this snapshot.

pending_push_extend_pc takes a snapshot of the result of merging the named proof context with the current proof context and then behaves just like *pending_push_merge_pc*.

The proof context must be in scope both at the time of the snapshot, and at the time of pushing on the stack.

This provides a method of being independent of changes to uncommitted proof contexts, or proof context deletions.

See Also *push_pc*

Errors

51010	*There is no proof context with key ?0*
51014	*Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified*

SML
```
val pending_reset_pc_database : unit -> unit -> unit;
```

Description This function, applied to () takes a snapshot of the proof context database, and returns a function that, if applied to () will restore the proof context database to the snapshot.

This function is particularly useful in initialising child databases, and in conjunction with *pending_reset_pc_stack* and *pending_reset_pc_evaluators*.

Note that a named proof context on the proof context stack is never taken as more than an echo of the item with that name (if any) of proof context database, and this function in particular, though not alone, is responsible for the possible differences.

SML
```
val pending_reset_pc_stack : unit -> unit -> unit;
```

Description This function, applied to () takes a snapshot of the proof context stack, and returns a function that, if applied to () will restore the proof context stack to the snapshot.

Uses This function is particularly useful in initialising child databases, and in conjunction with *pending_reset_pc_database* and *pending_reset_pc_evaluators*.

SML
```
val pending_reset_pc_evaluators : unit -> unit -> unit;
```

Description This function, applied to () takes a snapshot of the proof context evaluators (e.g. the one set by *pp'set_eval_ad_∃_vs_thms* , and returns a function that, if applied to () will restore the proof context evaluators to the snapshot.

Uses This function is particularly useful in initialising child databases, and in conjunction with *pending_reset_pc_database*, and *pending_reset_pc_stack*

SML
```
val pop_pc : unit -> unit;
```

Description This function unstacks the top of the proof context stack, and sets the current proof context of the system to it. There will always be a current proof context, though it may be the trivial "initial" proof context.

This function may make an out of scope proof context the current proof context.

See Also *push_pc*, *set_pc*, *push_merge_pcs*, *set_merge_pcs*

Errors
```
51001 The proof context stack is empty
```

SML
```
val pp'set_eval_ad_rw_net : (EQN_CXT -> CONV NET) -> unit;
val current_ad_rw_net : unit -> CONV NET;
```

Description These functions provide the interface to the initial conversion net for rewriting (see e.g. *rewrite_tac*) held in the application data of a proof context. The first sets the evaluator, the second extracts the field in the current proof context.

See Also *set_rw_eqn_cxt* for the associated user data.

Errors
```
51021 The current proof context was created in theory ?0 at a
      point now either not in scope, deleted or modified
```

SML

```
val pp'set_eval_ad_rw_canon : ((THM -> THM list) list -> (THM -> THM list))
        -> unit;
val current_ad_rw_canon : unit -> THM -> THM list;
```

Description These functions provide the interface to the canonicalisation function applied to rewriting theorems (see e.g. *rewrite_tac*) held in the application data of a proof context. The proof context is accessed after providing the theorem. The first sets the evaluator, the second extracts the field in the current proof context.

See Also *set_rw_canons* for the associated user data.

Errors

```
51021   The current proof context was created in theory ?0 at a
        point now either not in scope, deleted or modified
```

SML

```
val pp'set_eval_ad_st_conv : (EQN_CXT -> CONV) -> unit;
val current_ad_st_conv : unit -> CONV;
```

Description These functions provide the interface to the conversion for stripping theorems into the assumption list (see e.g. *strip_tac*) held in the application data of a proof context. The proof context is accessed before provision of a term. The first sets the evaluator, the second extracts the field in the current proof context.

See Also *set_st_conv* for the associated user data.

Errors

```
51021   The current proof context was created in theory ?0 at a
        point now either not in scope, deleted or modified
```

SML

```
val pp'set_eval_ad_sc_conv : (EQN_CXT -> CONV) -> unit;
val current_ad_sc_conv : unit -> CONV;
```

Description These functions provide the interface to the conversion for stripping goal conclusions (see e.g. *strip_tac*) held in the application data of a proof context. The proof context is accessed before provision of a term. The first sets the evaluator, the second extracts the field in the current proof context.

See Also *set_sg_conv* for the associated user data.

SML
```
val pp'set_eval_ad_nd_net :
        (string -> (TERM * (TERM -> THM)) list ->
                (TERM -> THM) NET) -> unit;
val current_ad_nd_net : string -> (TERM -> THM) NET;
```

Description These functions provide the interface to the additional dictionary of discrimination nets held in the application data of a list of proof contexts.

The application data is generated by taking, for each key in at least one of the dictionaries in the appropriate subfields of the proof context, the appended lists of all the entries for that key in any of the subfields of the proof context. To this is applied the evaluator set by *pp'set_eval_ad_nd_net* first applied to the dictionary key. The result is used as an entry, using the same dictionary key, in the resulting dictionary of nets. The default evaluator will just use *make_net* on each list of sources.

current_ad_nd_net key returns the net indexed by the key *key* in the current proof context. If no entry exists it returns the empty net *empty_net*. Note that the returned net can be viewed as something of type *EQN_CXT*, and made into a conversion by *eqn_cxt_conv*.

Uses For extending the proof context mechanisms. Though available to the end user, and indeed intended for use by the sophisticated user, the proof context mechanisms (as opposed to proof contexts) should be extended under ICL direction.

See Also *set_nd_entry* for the associated user data.

SML
```
val pp'set_eval_ad_cs_∃_convs : (CONV list -> CONV)
        -> unit;
val current_ad_cs_∃_conv : unit -> CONV;
```

Description These functions provide the interface to the existence prover for constant specifications (see *const_spec*) held in the application data of a proof context. The proof context is accessed before provision of a term. The first sets the evaluator, the second extracts the field in the current proof context.

See Also *set_cs_∃_rule* for the associated user data.

Errors
```
51015 No automated existence prover in the current proof context succeeds
51021 The current proof context was created in theory ?0 at a
      point now either not in scope, deleted or modified
```

SML
```
val pp'set_eval_ad_∃_cd_thms : (THM list ->
        (TERM list * int list * TYPE * (TERM list)list * THM) list) -> unit;
val current_ad_∃_cd_thms : unit ->
        (TERM list * int list * TYPE * (TERM list)list * THM) list;
```

Description These functions provide the interface to the clausal definition theorem information for the existence prover *prove_∃_conv*. See *evaluate_∃_cd_thms* for details upon the form of the information. The first sets the evaluator, the second extracts the field in the current proof context.

See Also *set_∃_cd_thms* for the associated user data.

Errors
```
51021 The current proof context was created in theory ?0 at a
      point now either not in scope, deleted or modified
```

SML

```
val pp'set_eval_ad_∃_vs_thms : ((string * (TERM list * THM)) list ->
        (string * (TERM list * THM)) list) -> unit;
val current_ad_∃_vs_thms : unit ->
        (string * (TERM list * THM)) list;
```

Description These functions provide the interface to the application data variable structure information for the existence prover *prove_∃_conv*. The first sets the evaluator, the second extracts the field in the current proof context.

See Also *set_∃_vs_thms* for user data.

Errors

```
51021 The current proof context was created in theory ?0 at a
      point now either not in scope, deleted or modified
```

SML

```
val prove_conv : THM list -> CONV;
```

Description This conversion is an automatic proof procedure appropriate to the current proof context.

At the point of applying this conversion to its theorems it will access the current setting of proof context field *pr_conv*, applying the result to the theorem list immediately, and then to the term when available (i.e. the result is partially evaluated with only the list of theorems).

Conversion

$$\frac{}{current\_ad\_pr\_conv\ ()\ thms\ \ulcorner t \urcorner} \quad \begin{array}{l} prove\_conv \\ thms\ \ulcorner t \urcorner \end{array}$$

See Also *PC_C1* to defer accessing the proof context until application to the term.

Errors

```
51021 The current proof context was created in theory ?0 at a
      point now either not in scope, deleted or modified
```

and as the proof context setting.

SML

```
val prove_rule : THM list -> TERM -> THM;
```

Description This rule is an automatic proof procedure appropriate to the current proof context.

At the point of applying this rule to its theorem list it will access the current setting of proof context field *pr_conv*, apply it to the theorem list immediately, and then to the term when available (i.e. the result is partially evaluated with only the list of theorems), and then, if the resulting theorem is '$\vdash term' \Leftrightarrow T$' (with no assumptions) where *term* is α-convertible to *term'*, then apply *⇔_t_elim*, and otherwise fail.

Rule

$$\frac{}{\vdash tm} \quad \begin{array}{l} prove\_rule \\ thms\ \ulcorner tm \urcorner \end{array}$$

See Also *pc_rule1* to defer accessing the proof context until application to the term.

Errors

```
51021 The current proof context was created in theory ?0 at a
      point now either not in scope, deleted or modified
51022 Result of applying conversion to ?0, which was ?1,
      not of form: '⊢ input ⇔ T'
```

and as the proof context setting.

SML

val **prove_∃_conv** : *CONV*;

Description This conversion is an automatic proof procedure for existential proofs, appropriate to the current proof context.

At the point of applying this conversion to a term it will access the current setting of proof context field *cs_∃_conv*, apply it to the theorem list, and then to the term.

The resulting theorem is not checked as having its L.H.S. being the input term.

Conversion

$$\frac{}{current\_ad\_cs\_\exists\_conv\ ()\ \ulcorner t \urcorner} \quad \begin{array}{l} prove\_\exists\_conv \\ \ulcorner t \urcorner \end{array}$$

Errors

51021 *The current proof context was created in theory ?0 at a point now either not in scope, deleted or modified*

and as the proof context setting.

SML

val **prove_∃_rule** : *TERM −> THM*;

Description This rule is an automatic proof procedure for existential proofs, appropriate to the current proof context.

At the point of applying this rule to a term *term* it will access the current setting of proof context field *cs_∃_conv*, apply it to the term, and then, if the resulting theorem is '$\vdash term' \Leftrightarrow T$' (with no assumptions) where *term* is α-convertible to $term'$, then apply *⇔_t_elim*, and otherwise fail.

Rule

$$\frac{}{\vdash tm} \quad \begin{array}{l} prove\_\exists\_rule \\ \ulcorner tm \urcorner \end{array}$$

Errors

51021 *The current proof context was created in theory ?0 at a point now either not in scope, deleted or modified*

51022 *Result of applying conversion to ?0, which was ?1, not of form: '⊢ input ⇔ T'*

and as the proof context setting.

SML

```
val push_extend_pcs : string list -> unit;
val set_extend_pcs : string list -> unit;
```

Description *push_extend_pcs* stacks the previous "current" proof context, and and then merges the proof contexts with the given keys into the current proof context. *set_extend_pcs* merges the proof contexts with the given keys into the previous current proof context without changing the stack.

Merged proof contexts upon the stack will have *current_ad_names* giving the list of names of the constituent proof contexts, singleton contexts will have singleton lists. The proof contexts used need not have been committed.

The *pr_conv*, *pr_tac* and *mmp_rule* fields take the value of the last proof context in the list that has the field set.

The current proof context is accessed by the functions prefixed *current_ad_*, and by *get_current-_pc*.

See Also *pop_pc*, *push_merge_pcs*, *set_merge_pcs*

Errors

```
51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a
      point now either not in scope, deleted or modified
51020 Must be at least one key in list
```

SML

```
val push_extend_pc : string -> unit;
val set_extend_pc : string -> unit;
```

Description *push_extend_pc* stacks the previous "current" proof context, and and then merges the proof context with the given key into the current proof context. *set_extend_pcs* merges the proof context with the given key into the current proof context without changing the stack.

Merged proof contexts upon the stack will have *current_ad_names* giving the list of names of the constituent proof contexts. The proof context used need not have been committed.

The *pr_conv*, *pr_tac* and *mpp_rule* fields take the value from the named proof context.

The current proof context is accessed by the functions prefixed *current_ad_*, and by *get_current-_pc*.

See Also *pop_pc*, *push_pc*, *set_pc*

Errors

```
51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a
      point now either not in scope, deleted or modified
51020 Must be at least one key in list
```

SML
```
val push_merge_pcs : string list -> unit;
val set_merge_pcs : string list -> unit;
```

Description *push_merge_pcs* stacks the previous "current" proof context, and and sets the current proof context of the system to the merge of the proof contexts with the given keys. *set_merge_pcs* discards the previous "current" proof context, and and sets the current proof context of the system to the merge of the proof contexts with the given keys. Merged proof contexts upon the stack will have *current_ad_names* giving the list of names of the constituent proof contexts, singleton contexts will have singleton lists. The proof contexts used need not have been committed.

The *pr_conv*, *pr_tac* and *mmp_rule* fields take the value of the last proof context in the list that has the field set.

The current proof context is accessed by the functions prefixed *current_ad_*, and by *get_current-_pc*.

See Also *pop_pc*, *push_pc*, *set_pc*

Errors
```
51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a
      point now either not in scope, deleted or modified
51020 Must be at least one key in list
```

SML
```
val push_pc : string -> unit;
val set_pc : string -> unit;
```

Description *push_pc* stacks the previous "current" proof context, and and sets the current proof context of the system to the proof context with the given key. *set_pc* discards the previous "current" proof context, and and sets the current proof context of the system to the proof context with the given key.

The current proof context is accessed by the functions prefixed *current_ad_*, and by *get_current-_pc*.

See Also *pending_push_pc*, *pop_pc*, *push_merge_pcs*, *set_merge_pcs*

Errors
```
51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a
      point now either not in scope, deleted or modified
```

SML

```
val set_cs_∃_convs : (CONV list) ->
        string -> unit;
val get_cs_∃_convs : string ->
        (((CONV list) * string) list);
```

Description These functions provide the interface to the existence provers for constant specifications (see *const_spec*) held in the user data of a proof context. Under the initial evaluator, the existence proving conversion supplied by *current_cs_∃_conv* will have each of the conversions tried, in the reverse order of their entry, being applied to the RHS of the result of the previous successful application, or the initial term to which the conversion was applied, until the RHS is ⌜T⌝, or no conversions remain.

Example

```
If get_cs_∃_convs of the current proof context returns

[([conv1, conv2],"pc1"),([conv3, conv4],"pc2")]

Then current_ad_cs_∃_conv  will return

conv4 AND_OR_C conv3 AND_OR_C conv2 AND_OR_C conv1
```

"setting" overwrites the subfield whose key is the proof context's name, "getting" returns the entire field (which pairs data with proof context names).

Errors

```
51010  There is no proof context with key ?0
51014  Proof context ?0 was created in theory ?1 at a
       point now either not in scope, deleted or modified
51016  Proof context ?0 has been committed
```

SML

```
val set_mmp_rule : (THM -> THM -> THM) -> string -> unit;
val get_mmp_rule : string -> (THM -> THM -> THM) OPT;
```

Description These functions provide the interface to the proof contexts for the matching modus ponens rule as used by tools such as *forward_chain_rule*. Note that setting overwrites all previous data in this field, including from merged in proof contexts. Merged proof contexts take their value for this field from the last proof context in the list that has the field set.

Errors

```
51010  There is no proof context with key ?0
51014  Proof context ?0 was created in theory ?1 at a
       point now either not in scope, deleted or modified
51016  Proof context ?0 has been committed
```

SML

```
val set_nd_entry : string -> (TERM * (TERM -> THM))list -> string -> unit;
val get_nd_entry : string -> string ->
        ((TERM * (TERM -> THM))list * string) list;
```

Description These functions provide the interface to the additional dictionary of sources for discrimination nets held in the user data of a proof context. The dictionary is actually a list of subfields of the proof context, indexed by source proof context name, each subfield being a dictionary in its own right. You "set" a single dictionary entry of the subfield indexed by the proof context's name (creating a new entry if necessary). You "get" the dictionaries for all the subfields.

set_nd_entry dict_key entry pc_name overwrites (or creates, if necessary) the proof context's name's subfield dictionary entry whose key is *dict_key* in the proof context *pc_name* with the value *entry*.

get_nd_entry dict_key pc_name returns the dictionary entries whose keys are *dict_key* from each of the subfields in the proof context *pc_name*, paired with the source proof context name, or an empty list if the entry is not present in the dictionaries of any of the subfields of that proof context.

Errors

51010	*There is no proof context with key ?0*
51014	*Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified*
51016	*Proof context ?0 has been committed*

SML

```
val set_pr_conv : (THM list -> CONV) -> string -> unit;
val get_pr_conv : string -> (THM list -> CONV);
val get_pr_conv1 : string -> (THM list -> CONV) OPT;
```

Description These functions provide the interface to the proof contexts for *prove_conv*. Note that setting overwrites all previous data in this field, including from merged in proof contexts. If the field has not been set, *get_pr_conv* returns a function mapping any list of theorems to *fail_conv* and *get_pr_conv1* returns *Nil*. Merged proof contexts take their value for this field from the last proof context in the list that has the field set.

Note that when using these functions that the standard rewriting functions (obvious candidates for inclusion in automatic proof) access the current proof context at the point of being given their theorem list argument.

Errors

51010	*There is no proof context with key ?0*
51014	*Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified*
51016	*Proof context ?0 has been committed*

SML

```
val set_pr_tac : (THM list -> TACTIC) -> string -> unit;
val get_pr_tac : string -> (THM list -> TACTIC);
val get_pr_tac1 : string -> (THM list -> TACTIC) OPT;
```

Description These functions provide the interface to the proof contexts for *prove_tac*. Note that setting overwrites all previous data in this field, including from merged in proof contexts. If the field has not been set, *get_pr_tac* returns a function mapping any list of theorems to *fail_tac* and *get_pr_tac1* returns *Nil*. Merged proof contexts take their value for this field from the last proof context in the list that has this field set.

When *asm_prove_tac* is applied to its theorem list argument the system will evaluate this by applying the value set by *set_pr_tac* for the current proof context to that argument. The provided values for *set_pr_tac* can interpret their theorem list arguments as they wish (e.g. as a set of rewrite theorems, or as theorems to resolve against) - no interpretation is forced upon this argument.

Note that when using these functions that the standard rewriting functions (obvious candidates for inclusion in automatic proof) access the current proof context at the point of being given their theorem list argument.

Errors

```
51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a
      point now either not in scope, deleted or modified
51016 Proof context ?0 has been committed
```

SML

```
val set_rw_canons : (THM -> THM list) list ->
      string -> unit;
val get_rw_canons : string -> ((THM -> THM list) list * string) list;
```

Description These functions provide the interface to the individual canonicalisation functions used to create the canonicalisation function applied to rewriting theorems (see e.g. *rewrite_tac*) held in the user data of a proof context.

“setting” overwrites the subfield whose key is the proof context’s name, “getting” returns the entire field (which pairs data with proof context names).

Errors

```
51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a
      point now either not in scope, deleted or modified
51016 Proof context ?0 has been committed
```

SML
```
val set_rw_eqm_rule : (THM -> TERM * CONV) -> string -> unit;
val get_rw_eqm_rule : string -> (THM -> TERM * CONV) OPT;
```

Description These functions provide the interface to the proof contexts for the equation matcher as used by the rewriting tools. Note that setting overwrites all previous data in this field, including from merged in proof contexts. Merged proof contexts take their value for this field from the last proof context in the list that has the field set.

Errors
```
51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a
      point now either not in scope, deleted or modified
51016 Proof context ?0 has been committed
```

SML
```
val set_rw_eqn_cxt : EQN_CXT -> string -> unit;
val get_rw_eqn_cxt : string -> (EQN_CXT * string) list;
val add_rw_thms : THM list -> string -> unit;
```

Description These functions provide the interface to the equational context for rewriting (see e.g. *rewrite_tac*) held in the user data of a proof context. "setting" overwrites the subfield whose key is the proof context's name, "getting" returns the entire field (which pairs data with proof context names). "adding" processes its theorems by first canonicalising according to the current proof context's canonicalisation function, and then with *thm_eqn_cxt* and then adds them into the subfield whose key is the proof context's name.

Errors
```
51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a
      point now either not in scope, deleted or modified
51016 Proof context ?0 has been committed
```

SML
```
val set_sc_eqn_cxt : EQN_CXT -> string -> unit;
val get_sc_eqn_cxt : string -> (EQN_CXT * string) list;
val add_sc_thms : THM list -> string -> unit;
```

Description These functions provide the interface to the equational context for stripping goal conclusions (see e.g. *strip_tac*) held in the user data of a proof context. "setting" overwrites the subfield whose key is the proof context's name, "getting" returns the entire field (which pairs data with proof context names). "adding" processes its theorems by first canonicalising according to the current proof context's canonicalisation function, and then with *thm_eqn_cxt* and then adds them into the subfield whose key is the proof context's name.

Errors
```
51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a
      point now either not in scope, deleted or modified
51016 Proof context ?0 has been committed
```

SML
```
val set_st_eqn_cxt : EQN_CXT -> string -> unit;
val get_st_eqn_cxt : string -> (EQN_CXT * string)list;
val add_st_thms : THM list -> string -> unit;
```

Description These functions provide the interface to the equational context for stripping theorems into the assumption list (see e.g. *strip_tac*) held in the user data of a proof context. "setting" overwrites the subfield whose key is the proof context's name, "getting" returns the entire field (which pairs data with proof context names). "adding" processes its theorems by first canonicalising according to the current proof context's canonicalisation function, and then with *thm_eqn_cxt* and then adds them into the subfield whose key is the proof context's name.

Errors

51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified
51016 Proof context ?0 has been committed

SML
```
val set_∃_cd_thms : THM list -> string -> unit;
val get_∃_cd_thms : string -> THM list;
val add_∃_cd_thms : THM list -> string -> unit;
```

Description These functions provide the interface to the unevaluated clausal definition theorems held for the existence prover *prove_∃_conv*. There are no subfields to this field, so "setting" overwrites the field with the proof context's name, "getting" returns the field. "adding" unions its theorem list with the proof contexts field.

See Also See *evaluate_∃_cd_thms* for details upon the form of the theorems.

Errors

51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified
51016 Proof context ?0 has been committed

SML
```
val set_∃_vs_thms : (string * (TERM list * THM)) list
        -> string -> unit;
val get_∃_vs_thms : string -> (string * (TERM list * THM)) list;
```

Description These functions provide the interface to the variable structure information for the existence prover *prove_∃_conv*. An individual entry in the list gives a method of handling an extended variable structure. It consists of the name of the constructor; a list of functions that extract each field of the constructor, and a theorem that states how the extraction functions extract from a data construction, and that the data constructor may be applied to the extracted values to regain the original value. For instance, for pairs the information is:

```
(",",
([⌜Fst⌝,⌜Snd⌝],
‘⊢ ∀ x y p •
        Fst (x, y) = x ∧ Snd (x, y) = y ∧
        (Fst p, Snd p) = p)‘))
```

There are no subfields to this field, so "setting" overwrites the field with the proof context's name, "getting" returns the field.

Errors
```
51010  There is no proof context with key ?0
51014  Proof context ?0 was created in theory ?1 at a
       point now either not in scope, deleted or modified
51016  Proof context ?0 has been committed
```

SML
```
val simple_ho_thm_eqn_cxt : THM -> (TERM * CONV);
```

Description This function is an equation matcher for use by the rewriting tools that uses higher-order matching. It transforms an equational theorem into a representation of a higher-order rewrite rule in a form suitable for inclusion in an an equational context (*EQN_CXT* q.v.)

```
thm_eqn_cxt ‘Γ ⊢ ∀ x1 ... • LHS = RHS‘ →
        (LHS′, simple_eq_match_conv1 ‘Γ ⊢ ∀ x1 ... • LHS = RHS‘)
```

Here the pattern term *LHS′* is derived from *LHS* by replacing linear patterns (see *simple_ho_match*) by variables of the same type.

The universal quantifiers must be over simple variables (not patterns) and the higher-order matching is done using *simple_ho_match*.

See Also *cthm_eqn_cxt* which canonicalises the theorem before transformation.

Errors
```
7095   ?0 is not of the form ‘Γ ⊢ ∀ x1 ... xn • u = v′ where ⌜xi⌝ are variables
```

SML

```
val thm_eqn_cxt : THM -> (TERM * CONV);
```

Description This function is a simple form of equation matcher for use by the rewriting tools. It transforms an equational theorem into a representation of a first-order rewrite rule in a form suitable for inclusion in an an equational context (*EQN_CXT* q.v.)

```
thm_eqn_cxt ‘Γ ⊢ ∀ x1 ... • LHS = RHS‘ →
        (LHS, simple_eq_match_conv1 ‘Γ ⊢ ∀ x1 ... • LHS = RHS‘)
```

The universal quantifiers must be over simple variables (not patterns).

See Also *cthm_eqn_cxt* which canonicalises the theorem before transformation.

Errors

```
7095   ?0 is not of the form ‘Γ ⊢ ∀ x1 ... xn • u = v′ where ⌜xi⌝ are variables
```

SML

```
signature ProofContexts1 = sig
```

Description This signature gives access to two functions used in supplying the first group of proof contexts. Proof contexts themselves have no entry in the signature, however the contexts provided are:

Component	Complete
'simple_abstractions	*predicates*
'paired_abstractions	*predicates1*
'propositions	*basic_hol*
'fun_ext	*basic_hol1*
'pair	*sets_ext*
'pair1	*hol*
'ℕ	*hol1*
'ℕ*_lit*	
'list	
'char	
'sum	
'one	
'combin	
'sets_alg	
'sets_ext	
'basic_prove_∃_conv	

SML

```
(* Proof Context: 'basic_prove_∃_conv *)
```

Description A component proof context that adds the function *basic_prove_∃_conv* as an automatic existence prover.

Contents Automatic proof procedures are respectively “always fail tactic”, “always fail conversion”, and *basic_prove_∃_conv*.

Usage Notes Requires theory “basic_hol”, intended to be combined into the merge of any component proof contexts that do not have their own special existence prover. It should usually be the first in the list of proof contexts to be merged together, so that other proof contexts may introduce pre-processors, and then the final default prover is invoked. This is because the standard application of the list of existence prover conversions is defined to be to apply them in a cumulative manner, in reverse order.

SML

|(∗ *Proof Context*: ′**simple_abstractions** ∗)

Description A component proof context for handling only simple abstractions in stripping and canonicalisation.

Contents Rewriting:

Stripping theorems:

|*simple_¬_in_conv*

Stripping conclusions:

|*simple_¬_in_conv*

Rewriting canonicalisation:

|*simple_∀_rewrite_canon*, *simple_¬_rewrite_canon*

Automatic proof procedures are respectively *basic_prove_tac*, *basic_prove_conv* and no existence prover.

Usage Notes Not to be used with proof context "′paired_abstractions" as their "domains" overlap. It requires theory *basic_hol*.

SML

|(∗ *Proof Context*: ′**paired_abstractions** ∗)

Description A component proof context for handling simple and paired abstractions in stripping and canonicalisation.

Contents Rewriting:

|*β_conv*

Stripping theorems:

|*¬_in_conv*, *∃₁_conv*,
|*∀_uncurry_conv*, *∃_uncurry_conv*

Stripping conclusions:

|*¬_in_conv*, *∀_uncurry_conv*

Rewriting canonicalisation:

|*∀_rewrite_canon*, *¬_rewrite_canon*

Automatic proof procedures are respectively *basic_prove_tac*, *basic_prove_conv* and no existence prover.

Usage Notes Not to be used with proof context "′simple_abstractions", as their "domains" overlap. It requires theory *basic_hol*.

SML

```
(∗ Proof Context: ′propositions ∗)
```

Description A component proof context for reasoning about propositions.

Contents Rewriting:

$eq\_rewrite\_thm$, $\Leftrightarrow\_rewrite\_thm$, $\neg\_rewrite\_thm$,
$\wedge\_rewrite\_thm$, $\vee\_rewrite\_thm$, $\Rightarrow\_rewrite\_thm$,
$if\_rewrite\_thm$, $\forall\_rewrite\_thm$, $\exists\_rewrite\_thm$,
$\beta\_rewrite\_thm$, $simple\_\beta\_conv$

Stripping theorems:

$\Rightarrow\_thm$, $\Leftrightarrow\_thm$, $simple\_\exists_1\_conv$,
‘$\vdash \forall\ x \bullet ((x = x) \Leftrightarrow T)$‘,
‘$\vdash \forall\ x \bullet (\neg(x = x) \Leftrightarrow F)$‘,
‘$\vdash \forall\ a\ t1\ t2 \bullet (if\ a\ then\ t1\ else\ t2) \Leftrightarrow (a \Rightarrow t1) \wedge (\neg\ a \Rightarrow t2)$‘

Note these are intended to be used with ($simple\_$) $\neg\_in\_conv$ from "′paired_abstractions" or "′simple_abstractions", which covers the cases of an outermost ¬ for each operator.

Stripping conclusions:

$\Leftrightarrow\_thm$,
‘$\vdash \forall\ x \bullet ((x = x) \Leftrightarrow T)$‘,
‘$\vdash \forall\ x \bullet (\neg(x = x) \Leftrightarrow F)$‘,
‘$\vdash \forall\ a\ t1\ t2 \bullet (if\ a\ then\ t1\ else\ t2) \Leftrightarrow (a \Rightarrow t1) \wedge (\neg\ a \Rightarrow t2)$‘
‘$\vdash \forall a\ b \bullet (a \vee \neg b) \Leftrightarrow (b \Rightarrow a)$‘
‘$\vdash \forall\ a\ b \bullet \neg\ a \vee b \Leftrightarrow a \Rightarrow b$‘
‘$\vdash \forall\ a\ b \bullet\ a \vee b \Leftrightarrow \neg\ a \Rightarrow b$‘

Note that the above are intended to be used in combination with ($simple\_$) $\neg\_in\_conv$ from "′paired_abstractions" or "′simple_abstractions", which covers the cases of an outermost ¬ for each operator.

Rewriting canonicalisation:

$\wedge\_rewrite\_canon$, $f\_rewrite\_canon$

Automatic proof procedures are respectively $taut\_tac$, $taut\_conv$ and $basic\_prove\_\exists\_conv$.

Usage Notes Usually used in conjunction with "′paired_abstractions" or "′simple_abstractions", requires theory $basic\_hol$.

SML

|(∗ *Proof Context*: **′fun_ext** ∗)

Description A component proof context for adding reasoning using functional extensionality.

Contents Rewriting:

|*ext_thm*

Stripping theorems:

|*ext_thm*

Stripping conclusions:

|*ext_thm*

Rewriting canonicalisation:

Automatic proof procedures are, respectively, *taut_tac*, *taut_conv* and *basic_prove_∃_conv*.

Usage Notes Normally used in conjunction with "′propositions", requires theory *basic_hol*.

SML

|(∗ *Proof Context*: **predicates** ∗)

Description A "mild" complete proof context for reasoning about the predicate calculus, including paired abstractions.

Contents Proof contexts "′basic_prove_∃_conv", "′paired_abstractions" and "′propositions".

Automatic proof procedures are respectively *basic_prove_tac*, *basic_prove_conv* and *basic_prove_∃_conv* (merged in from the proof context of the same name).

Usage Notes Requires theory *basic_hol*.

SML

|(∗ *Proof Context*: **predicates1** ∗)

Description An "aggressive" complete proof context for reasoning about the predicate calculus, including paired abstractions and functional extensionality.

Contents Proof contexts "′basic_prove_∃_conv", "′paired_abstractions", "′propositions" and "′fun_ext".

Automatic proof procedures are, respectively, *basic_prove_tac*, *basic_prove_conv* and *basic_prove_∃_conv* (merged in from the proof context of the same name).

Usage Notes Requires theory *basic_hol*.

SML
```
(* Proof Context: 'pair *)
```

Description A “mild” component proof context for theory *pair*.

Contents Rewriting (selected from *pair_clauses*):

$$
\begin{array}{l}
`\vdash \forall\ x\ y\ a\ b\ p\ fu\ fc \\
\quad \bullet\ Fst\ (x,\ y) = x \\
\quad\quad \land\ Snd\ (x,\ y) = y \\
\quad\quad \land\ ((a,\ b) = (x,\ y) \Leftrightarrow a = x \land b = y) \\
\quad\quad \land\ (Fst\ p,\ Snd\ p) = p \\
\quad\quad \land\ Curry\ fc\ x\ y = fc\ (x,\ y) \\
\quad\quad \land\ Uncurry\ fu\ (x,\ y) = fu\ x\ y \\
\quad\quad \land\ Uncurry\ fu\ p = fu\ (Fst\ p)\ (Snd\ p)`
\end{array}
$$

Stripping theorems:

$$`\vdash \forall\ a\ b\ x\ y \bullet ((a,\ b) = (x,\ y) \Leftrightarrow a = x \land b = y)`$$

Stripping conclusions:

$$`\vdash \forall\ a\ b\ x\ y \bullet ((a,\ b) = (x,\ y) \Leftrightarrow a = x \land b = y)`$$

Existential variable structures:

$$
\begin{array}{l}
`\vdash \forall\ x\ y\ p \bullet \\
\quad\quad\quad Fst\ (x,\ y) = x\ \land \\
\quad\quad\quad Snd\ (x,\ y) = y\ \land \\
\quad\quad\quad (Fst\ p,\ Snd\ p) = p`
\end{array}
$$

Automatic proof procedures are respectively *basic_prove_tac*, *basic_prove_conv* and no existence prover.

Usage Notes Requires theory *basic_hol*.

SML

(∗ *Proof Context*: ′**pair1** ∗)

Description An "aggressive" component proof context for theory *pair*.

Contents Rewriting:

$‘\vdash \forall\ a\ b\ p$
$\quad \bullet ((a,\ b) = p \Leftrightarrow a = Fst\ p \wedge b = Snd\ p)$
$\quad \wedge (p = (a,\ b) \Leftrightarrow Fst\ p = a \wedge Snd\ p = b)‘$

Stripping theorems (selected from *pair_clauses*):

$‘\vdash \forall\ a\ b\ p$
$\quad \bullet ((a,\ b) = p \Leftrightarrow a = Fst\ p \wedge b = Snd\ p)$
$\quad \wedge (p = (a,\ b) \Leftrightarrow Fst\ p = a \wedge Snd\ p = b)‘$

Stripping conclusions:

$‘\vdash \forall\ a\ b\ p$
$\quad \bullet ((a,\ b) = p \Leftrightarrow a = Fst\ p \wedge b = Snd\ p)$
$\quad \wedge (p = (a,\ b) \Leftrightarrow Fst\ p = a \wedge Snd\ p = b)‘$

Existential variable structures:

Automatic proof procedures are respectively *basic_prove_tac*, *basic_prove_conv* and no existence prover.

Usage Notes Requires theory *basic_hol*, expected to be used in combination with "′pair".

SML

(∗ *Proof Context*: ′ℕ ∗)

Description A "mild" component proof context for theory ℕ.

Contents Rewriting:

≥_def, *greater_def*, *plus_clauses*, *times_clauses*,
≤_clauses, *less_clauses*, *minus_clauses*

Stripping theorems:

≥_def, *greater_def*, *plus_clauses*, *times_clauses*,
≤_clauses, *less_clauses*, *minus_clauses*,
and all boolean equations also pushed through $¬$

Stripping conclusions:

≥_def, *greater_def*, *plus_clauses*, *times_clauses*,
≤_clauses, *less_clauses*, *minus_clauses*,
and all boolean equations also pushed through $¬$

Existential clausal definition theorems:

prim_rec_thm

Automatic proof procedures are respectively *basic_prove_tac*, *basic_prove_conv* and no existence prover.

Usage Notes Requires theory *basic_hol*.

SML

```
(∗ Proof Context: 'ℕ_lit ∗)
```

Description A component proof context for theory ℕ, that will, e.g., evaluate any arithmetic expression involving only numeric literals and certain arithmetic operators, namely $+$, $*$, $-$, *Div*, *Mod*, $\leq$, $<$, $>$, $\geq$, *and* $=$.

Contents Rewriting:

```
plus_conv, times_conv, minus_conv, div_conv,
mod_conv, ≤_conv, less_conv, greater_conv,
≥_conv, ℕ_eq_conv
```

Stripping theorems:

```
≤_conv, less_conv, greater_conv,
≥_conv, ℕ_eq_conv
```

Stripping conclusions:

```
≤_conv, less_conv, greater_conv,
≥_conv, ℕ_eq_conv
```

Existential clausal definition theorems:

Automatic proof procedures are respectively *basic_prove_tac*, *basic_prove_conv* and no existence prover.

Usage Notes Requires theory *basic_hol*, expected to be used with proof context "'ℕ". It is separated from it as spotting the application of the conversions is time consuming, and may be known to be irrelevant.

SML
```
(* Proof Context: 'list *)
```

Description A component proof context for the theory *list*.

Contents Rewriting:

```
list_clauses
```

Stripping theorems:

```
⌜ ⊢ ∀ x1 x2 list1 list2
    • ¬ Cons x1 list1 = []
     ∧ ¬ [] = Cons x1 list1
     ∧ (Cons x1 list1 = Cons x2 list2 ⇔ x1 = x2 ∧ list1 = list2)⌝
```

Stripping conclusions:

```
⌜ ⊢ ∀ x1 x2 list1 list2
    • ¬ Cons x1 list1 = []
     ∧ ¬ [] = Cons x1 list1
     ∧ (Cons x1 list1 = Cons x2 list2 ⇔ x1 = x2 ∧ list1 = list2)⌝
```

Existential clausal definition theorems:

```
list_prim_rec_thm
```

Automatic proof procedures are respectively *basic_prove_tac*, *basic_prove_conv* and no existence prover.

Usage Notes Requires theory *list*.

SML
```
(* Proof Context: 'char *)
```

Description A component proof context for theory *char*, for reasoning about character and string literals.

Contents Rewriting:

```
char_eq_conv, string_eq_conv
```

Stripping theorems:

```
char_eq_conv, string_eq_conv
```

Stripping conclusions:

```
char_eq_conv, string_eq_conv
```

Automatic proof procedures are respectively *basic_prove_tac*, *basic_prove_conv* and an existence prover preprocessor that rewrites with ⊢ "" = [] which assists using list's primitive induction on strings.

Usage Notes Requires theory *basic_hol*.

SML
```
(* Proof Context: basic_hol *)
```

Description A "mild" complete proof context for the ancestors of theory *basic_hol*.

Contents Proof contexts "predicates", "'pair, "'ℕ", "'list", and "'char". Automatic proof procedures are respectively *basic_prove_tac*, *basic_prove_conv* and *basic_prove_∃_conv* (merged in from the proof context of the same name).

Usage Notes Requires theory *basic_hol*.

SML
```
(* Proof Context: basic_hol1 *)
```

Description An "aggressive" complete proof context for the ancestors of theory *basic_hol*.

Contents Proof contexts "predicates1", "'pair", "'pair1", "'ℕ", "'ℕ_lit", "'list", and "'char".

Automatic proof procedures are respectively *basic_prove_tac*, *basic_prove_conv* and *basic_prove-_∃_conv* (merged in from the proof context of the same name).

Usage Notes Requires theory *basic_hol*.

SML
```
(* Proof Context: 'mmp1 *)
```

Description A component proof context with the matching modus ponens rule set to *⇒_match_mp_rule1*. All other fields are empty.

Usage Notes This makes forward chaining work as in releases prior to 2.9.1 (so that bound variables that are not constrained by the pattern matching are specialised to themselves).

SML
```
(* Proof Context: 'mmp2 *)
```

Description A component proof context with the matching modus ponens rule set to *⇒_match_mp_rule2*. All other fields are empty.

Usage Notes Use this to ensure the default behaviour in forward chaining (so that bound variables that are not constrained by the pattern matching are specialised with new names as necessary to avoid variable capture).

SML
```
(* Proof Context: 'sho_rw *)
```

Description A component proof context with the equation matching rule set to *simple_higher_order_thm_eqn_cxt*. All other fields are empty.

Usage Notes With this proof context, rewriting treats the rewriting theorems as higher order rewrite rules. For example, rewriting with the theorem *prenex_clauses* (q.v.) will convert a term into prenex normal form.

SML

```
(∗ Proof Context: ′sum ∗)
```

Description A “mild” component proof context for theory *sum*.

Contents Rewriting:

```
‘ ⊢ ∀ x1 x2 y1 y2 z
    • (InL x1 = InL x2 ⇔ x1 = x2)
      ∧ (InR y1 = InR y2 ⇔ y1 = y2)
      ∧ ¬ InL x1 = InR y1
      ∧ ¬ InR y1 = InL x1
      ∧ OutL (InL x1) = x1
      ∧ OutR (InR y1) = y1‘
      ∧ IsL(InL x1) ∧ IsR(InR y1)
      ∧ ¬ IsL(InR y1) ∧ ¬ IsR(InL x1)
```

Stripping theorems:

```
‘ ⊢ ∀ x1 x2 y1 y2 z
    • (InL x1 = InL x2 ⇔ x1 = x2)
      ∧ (InR y1 = InR y2 ⇔ y1 = y2)
      ∧ ¬ InL x1 = InR y1
      ∧ ¬ InR y1 = InL x1‘
      ∧ IsL(InL x1) ∧ IsR(InR y1)
      ∧ ¬ IsL(InR y1) ∧ ¬ IsR(InL x1)
```

Stripping conclusions:

```
‘ ⊢ ∀ x1 x2 y1 y2 z
    • (InL x1 = InL x2 ⇔ x1 = x2)
      ∧ (InR y1 = InR y2 ⇔ y1 = y2)
      ∧ ¬ InL x1 = InR y1
      ∧ ¬ InR y1 = InL x1‘
      ∧ IsL(InL x1) ∧ IsR(InR y1)
      ∧ ¬ IsL(InR y1) ∧ ¬ IsR(InL x1)
```

Existential clausal definition theorems:

```
‘⊢ ∀ f g• ∃₁ h• (∀ x• h (InL x) = f x) ∧ (∀ x• h (InR x) = g x)‘
```

Automatic proof procedures are respectively *basic_prove_tac*, *basic_prove_conv* and no existence prover.

Usage Notes Requires theory *sum*.

SML

(∗ *Proof Context*: ′**one** ∗)

Description A component proof context for theory *one*

Contents Rewriting (these both have the problem that their discrimination net entry will match anything):

one_def, *one_fns_thm*

Stripping theorems:

$‘ \vdash \forall\ x\ y : ONE \bullet (x = y) \Leftrightarrow T‘$
$‘ \vdash \forall\ x\ y : {}'a \to ONE \bullet (x = y) \Leftrightarrow T‘$
and through ¬

Stripping conclusions:

$‘ \vdash \forall\ x\ y : ONE \bullet (x = y) \Leftrightarrow T‘$
$‘ \vdash \forall\ x\ y : {}'a \to ONE \bullet (x = y) \Leftrightarrow T‘$
and through ¬

Automatic proof procedures are respectively *basic_prove_tac*, *basic_prove_conv* and no existence prover.

Usage Notes Requires theory *one*. As when entered into the rewriting net the rewriting theorems will match any term presented to the net, this proof context will slow down rewriting.

SML

(∗ *Proof Context*: ′**combin** ∗)

Description A component proof context for theory *combin*

Contents Rewriting:

comb_i_def, *comb_k_def*, *o_def*, *o_i_thm*

Stripping theorems:

Stripping conclusions:

Automatic proof procedures are, respectively, *basic_prove_tac*, *basic_prove_conv* and no existence prover.

Usage Notes Requires theory *combin*.

SML

```
(∗ Proof Context: ′sets_alg ∗)
```

Description A "mild" component proof context for theory *set*.

Contents Rewriting:

```
∈_comp_conv, ∈_enum_set_conv, complement_clauses,
∪_clauses, ∩_clauses, set_dif_clauses, ⊖_clauses,
⊆_clauses, ⊂_clauses, ⋃_clauses,
⋂_clauses, ℙ_clauses
‘ ⊢ ∀ x y
        • ¬ x ∈ {}
        ∧ x ∈ Universe
        ∧ (x ∈ {y} ⇔ x = y)‘
```

Stripping theorems:

```
∈_comp_conv, ∈_enum_set_conv, ∈_in_clauses
⊆_clauses, ⊂_clauses
plus these all pushed in through ¬
```

Stripping conclusions:

```
∈_comp_conv, ∈_enum_set_conv, ∈_in_clauses
⊆_clauses, ⊂_clauses
plus these all pushed in through ¬
```

Automatic proof procedures are respectively *basic_prove_tac*, *basic_prove_conv* and the existence prover preprocessor:

```
TOP_MAP_C (all_∃_uncurry_conv AND_OR_C sets_simple_∃_conv)
```

. The preprocessor causes set membership (∈) to be treated as function application in some cases.

Usage Notes Should not be used with proof context "′sets_ext", requires theory *sets*.

SML
```
(* Proof Context: 'sets_ext *)
```

Description A component proof context for theory *set*, "aggressively" using the extensionality of sets.

Contents Rewriting:

```
∈_comp_conv, ∈_enum_set_conv, ∈_in_clauses, sets_ext_clauses
```

Stripping theorems:

```
∈_comp_conv, ∈_enum_set_conv, ∈_in_clauses, sets_ext_clauses
plus these all pushed in through ¬
```

Stripping conclusions:

```
∈_comp_conv, ∈_enum_set_conv, ∈_in_clauses, sets_ext_clauses
plus these all pushed in through ¬
```

Automatic proof procedures are respectively *basic_prove_tac*, *basic_prove_conv* and the existence prover preprocessor:

```
TOP_MAP_C (all_∃_uncurry_conv AND_OR_C sets_simple_∃_conv)
```

The preprocessor causes set membership (∈) to be treated as function application in some cases.

Usage Notes Should not be used with proof context "'sets_alg", requires theory *sets*.

SML
```
(* Proof Context: sets_ext *)
```

Description A complete proof context for reasoning about sets within the predicate calculus, "aggressively" using the extensionality of sets.

Contents Proof contexts "'sets_ext" and "predicates".

Usage Notes Requires theory *sets*. The proof context "'sets_ext1" offers a much more useful treatment of sets of pairs.

SML
(∗ *Proof Context*: **′sets_ext1** ∗)

Description A component proof context for theory *set*, including sets of pairs, "aggressively" using the extensionality of sets.

Contents Rewriting:

$\in$_*comp_conv*, $\in$_*enum_set_conv*, $\in$_*in_clauses*,
sets_eq_conv, $\subseteq$_*conv*, $\subset$_*conv*

Stripping theorems:

$\in$_*comp_conv*, $\in$_*enum_set_conv*, $\in$_*in_clauses*,
sets_eq_conv, $\subseteq$_*conv*, $\subset$_*conv*
plus these all pushed in through $\neg$

Stripping conclusions:

$\in$_*comp_conv*, $\in$_*enum_set_conv*, $\in$_*in_clauses*,
sets_eq_conv, $\subseteq$_*conv*, $\subset$_*conv*
plus these all pushed in through $\neg$

Automatic proof procedures are respectively *basic_prove_tac*, *basic_prove_conv* and the existence prover preprocessor:

TOP_MAP_C (*all_∃_uncurry_conv AND_OR_C sets_simple_∃_conv*)

The preprocessor causes set membership ($\in$) to be treated as function application in some cases.

Usage Notes Should not be used with proof context "′sets_alg", requires theory *sets*.

SML
(∗ *Proof Context*: **sets_ext** ∗)

Description A complete proof context for reasoning about sets within the predicate calculus, "aggressively" using the extensionality of sets.

Contents Proof contexts "′sets_ext" and "predicates".

Usage Notes Requires theory *sets*.

SML
(∗ *Proof Context*: **sets_ext1** ∗)

Description A complete proof context for reasoning about sets, including sets of pairs, within the predicate calculus, "aggressively" using the extensionality of sets.

Contents Proof contexts "′sets_ext1" and "predicates".

Usage Notes Requires theory *sets*. The proof context "sets_ext1" offers a much more useful treatment of sets of pairs.

SML
(∗ *Proof Context*: **hol** ∗)

Description A "mild" complete proof context for the ancestors of theory *hol*

Contents Proof contexts "basic_hol", "′sum", "′combin", and "′sets_alg".

Automatic proof procedures are respectively *basic_prove_tac*, *basic_prove_conv* and *basic_prove-_∃_conv* (merged in from the proof context of the same name).

Usage Notes Requires theory *hol*.

SML
```
(∗ Proof Context: hol1 ∗)
```

Description An "aggressive" complete proof context for the ancestors of theory *hol*.

Contents Proof contexts "basic_hol1", "′one", "′sum", "′combin", and "′sets_ext".

Automatic proof procedures are respectively *basic_prove_tac*, *basic_prove_conv* and *basic_prove_∃_conv* (merged in from the proof context of the same name).

Usage Notes Requires theory *hol*. The proof context *hol2* offers a more useful treatment of sets of pairs.

SML
```
(∗ Proof Context: hol2 ∗)
```

Description An "aggressive" complete proof context for the ancestors of theory *hol*.

Contents Proof contexts "basic_hol1", "′one", "′sum", "′combin", and "′sets_ext1".

Automatic proof procedures are respectively *basic_prove_tac*, *basic_prove_conv* and *basic_prove_∃_conv* (merged in from the proof context of the same name).

Usage Notes Requires theory *hol*.

SML

```
val basic_prove_conv : THM list -> CONV;
```

Description This is the conversion used for the automatic proof conversion (*pr_tac* field) of most supplied proof contexts, and is a reasonable, general-purpose, automatic proof conversion. It will either prove the theorem with the given conclusion, or fail.

In summary it will:

1. Set the term as the goal of the subgoal package.
2. Attempt to rewrite the term with the current default rewrite rules and given theorems.
3. Repeatedly apply *strip_tac* to the goal.
4. Try *all_var_elim_asm_tac* to do variable elimination.
5. Attempt to prove the resulting goals with resolution for up to 3 resolution steps, with goal's negated conclusion as a resolvant that must be used, and the assumptions as possible other resolvants. This has no effect on any resulting goal if it is unsolved.
6. Attempt to prove the resulting goals with resolution for up to 3 resolution steps amongst just the assumptions. This has no effect on any resulting goal if it is unsolved.
7. If the proof is successful, return ⊢ *term* ⇔ *T* and otherwise fail.

Note that in the stripping step may result in more than one subgoal, and thus the plural "resulting goals".

Under the current interface to resolution this equivalent to:

```
fun basic_prove_conv thms tm =
⇔_t_intro (
tac_proof(([],tm),
        TRY_T (rewrite_tac thms)  THEN
        REPEAT strip_tac THEN_TRY
        (basic_res_tac2 3 [⊢ ∀ x • x = x]
         ORELSE_T basic_res_tac3 3 [⊢ ∀ x • x = x]))
)
```

In the implementation however, partial evaluation with just the theorems is allowed.

Errors

```
76001 Could not prove theorem with conclusion ?0
```

SML
```
val basic_prove_tac : THM list -> TACTIC;
```

Description This is the tactic used for the automated proof tactic (the *pr_tac* field) of most supplied proof contexts, and is a reasonable, general-purpose, automatic proof tactic.

In summary it will:

1. Try *all_var_elim_asm_tac* to do variable elimination.
2. Extract the assumption list, rewrite each extracted assumption with the current default rewrite rules and given theorems, and strip the results back into the assumption list.
3. Attempt to rewrite the resulting goal's conclusions with the current default rewrite rules and given theorems.
4. Again try *all_var_elim_asm_tac* to do variable elimination.
5. Repeatedly apply *strip_tac* to the conclusions of the resulting goals.
6. Attempt to prove each of the resulting goals with resolution for up to 3 resolution steps, with goal's negated conclusion as a resolvant that must be used, and the assumptions as possible other resolvants. This has no effect on any resulting goal if it is unsolved.
7. Attempt to prove each of the resulting goals with resolution for up to 3 resolution steps amongst just the assumptions. This has no effect on any resulting goal if it is unsolved.

Note that either stripping step may result in more than one subgoal, and thus the plural "resulting goals".

Under the current interface to resolution this is

```
fun basic_prove_tac thms =
TRY_T all_var_elim_asm_tac THEN
DROP_ASMS_T (MAP_EVERY (strip_asm_tac o rewrite_rule thms) o rev) THEN
(TRY_T (rewrite_tac thms)) THEN
TRY_T all_var_elim_asm_tac THEN
REPEAT strip_tac THEN_TRY
(basic_res_tac2 3 [⊢ ∀ x • x = x]
 ORELSE_T basic_res_tac3 3 [⊢ ∀ x • x = x])
```

THEORIES

9.1 Theory Listings

This section contains the listings of each theory.

9.1.1 The Theory basic_hol

9.1.1.1 Parents

char

9.1.1.2 Children

sets *combin one*

9.1.1.3 Notes

This theory is a cache theory; its contents have not been listed.

9.1.2 The Theory bin_rel

9.1.2.1 Parents

hol

9.1.2.2 Children

fun_rel

9.1.2.3 Constants

$\$\mapsto$	$'a \to 'b \to 'a \times 'b$
$\$\times$	$'a\ SET \to 'b\ SET \to ('a \times 'b)\ SET$
$\$\leftrightarrow$	$'a\ SET \to 'b\ SET \to ('a \times 'b)\ SET\ SET$
Dom	$('a \times 'b)\ SET \to 'a\ SET$
Ran	$('a \times 'b)\ SET \to 'b\ SET$
Id	$'a\ SET \to ('a \times 'a)\ SET$
Graph	$('a \to 'b) \to ('a \times 'b)\ SET$
$\$\fatsemi$	$('a \to 'b) \to ('b \to 'c) \to 'a \to 'c$
$\$$**R_$\fatsemi$_R**	$('a \times 'b)\ SET \to ('b \times 'c)\ SET \to ('a \times 'c)\ SET$
$\$$**R_o_R**	$('b \times 'c)\ SET \to ('a \times 'b)\ SET \to ('a \times 'c)\ SET$
$\$\lhd$	$'a\ SET \to ('a \times 'b)\ SET \to ('a \times 'b)\ SET$
$\$\rhd$	$('a \times 'b)\ SET \to 'b\ SET \to ('a \times 'b)\ SET$
$\$\ntriangleleft$	$'a\ SET \to ('a \times 'b)\ SET \to ('a \times 'b)\ SET$
$\$\ntriangleright$	$('a \times 'b)\ SET \to 'b\ SET \to ('a \times 'b)\ SET$
InvRel	$('a \times 'b)\ SET \to ('b \times 'a)\ SET$
$\$$**Image**	$('a \times 'b)\ SET \to 'a\ SET \to 'b\ SET$
Reflexive	$('a \times 'a)\ SET\ SET$
Symmetric	$('a \times 'a)\ SET\ SET$
Transitive	$('a \times 'a)\ SET\ SET$
Injective	$('a \times 'b)\ SET\ SET$
Surjective	$'b\ SET \to ('a \times 'b)\ SET\ SET$
Total	$'a\ SET \to ('a \times 'b)\ SET\ SET$
Functional	$('a \times 'b)\ SET\ SET$
$\$\oplus$	$('a \times 'b)\ SET \to ('a \times 'b)\ SET \to ('a \times 'b)\ SET$
$\$^{+}$	$('a \times 'a)\ SET \to ('a \times 'a)\ SET$
$\$^{*}$	$('a \times 'a)\ SET \to ('a \times 'a)\ SET$
RelCombine	$('a \times 'b)\ SET \to ('a \times 'c)\ SET \to ('a \times 'b \times 'c)\ SET$

9.1.2.4 Aliases

$\fatsemi$	$\$R\_\fatsemi\_R : ('b \times 'a)\ SET \to ('a \times 'c)\ SET \to ('b \times 'c)\ SET$
o	$\$R\_o\_R : ('a \times 'c)\ SET \to ('b \times 'a)\ SET \to ('b \times 'c)\ SET$
$\sim$	$InvRel : ('b \times 'a)\ SET \to ('a \times 'b)\ SET$

9.1.2.5 Type Abbreviations

$'a\ \mathbb{P}$	$'a\ SET$
$'a \leftrightarrow 'b$	$('a \times 'b)\ SET$

9.1.2.6 Fixity

Right Infix 240: **R_o_R** **R_⨾_R** ⩥ ▷ ↔ ⊕ ⨾ ⩤ ◁

Right Infix 280: **Image**

Right Infix 300: ↦

Postfix 300: * + ~

9.1.2.7 Definitions

↦	$\vdash \$\!\mapsto\; = \$,$
×	$\vdash \forall\ x\ y\bullet\ (x\ \times\ y) = \{(v,\ w) \mid v \in x \wedge w \in y\}$
↔	$\vdash \forall\ x\ y\bullet\ x \leftrightarrow y = \mathbb{P}\ (x\ \times\ y)$
Dom	$\vdash \forall\ r\bullet\ Dom\ r = \{x \mid \exists\ y\bullet\ (x,\ y) \in r\}$
Ran	$\vdash \forall\ r\bullet\ Ran\ r = \{y \mid \exists\ x\bullet\ (x,\ y) \in r\}$
Id	$\vdash \forall\ s\bullet\ Id\ s = \{(x,\ y) \mid x = y \wedge x \in s\}$
Graph	$\vdash \forall\ f\bullet\ Graph\ f = \{(x,\ y) \mid y = f\ x\}$
⨾	$\vdash \forall\ f\ g\bullet\ f\ ⨾\ g = g\ o\ f$
R_⨾_R	$\vdash \forall\ r\ s\bullet\ r\ ⨾\ s = \{(x,\ z) \mid \exists\ y\bullet\ (x,\ y) \in r \wedge (y,\ z) \in s\}$
R_o_R	$\vdash \forall\ r\ s\bullet\ r\ o\ s = s\ ⨾\ r$
◁	$\vdash \forall\ a\ r\bullet\ a \lhd r = \{(x,\ y) \mid x \in a \wedge (x,\ y) \in r\}$
▷	$\vdash \forall\ a\ r\bullet\ r \rhd a = \{(x,\ y) \mid y \in a \wedge (x,\ y) \in r\}$
⩤	$\vdash \forall\ a\ r\bullet\ a ⩤ r = \{(x,\ y) \mid \neg\ x \in a \wedge (x,\ y) \in r\}$
⩥	$\vdash \forall\ a\ r\bullet\ r ⩥ a = \{(x,\ y) \mid \neg\ y \in a \wedge (x,\ y) \in r\}$
InvRel	$\vdash \forall\ r\bullet\ r\ ^{\sim} = \{(x,\ y) \mid (y,\ x) \in r\}$
Image	$\vdash \forall\ r\ s\bullet\ r\ Image\ s = \{y \mid \exists\ x\bullet\ x \in s \wedge (x,\ y) \in r\}$
Reflexive	$\vdash Reflexive = \{r \mid \forall\ x\bullet\ (x,\ x) \in r\}$
Symmetric	$\vdash Symmetric = \{r \mid \forall\ x\ y\bullet\ (x,\ y) \in r \Rightarrow (y,\ x) \in r\}$
Transitive	$\vdash Transitive$ $= \{r \mid \forall\ x\ y\ z\bullet\ (x,\ y) \in r \wedge (y,\ z) \in r \Rightarrow (x,\ z) \in r\}$
Injective	$\vdash Injective$ $= \{r \mid \forall\ x\ y\ z\bullet\ (x,\ z) \in r \wedge (y,\ z) \in r \Rightarrow x = y\}$
Surjective	$\vdash \forall\ s\bullet\ Surjective\ s = \{r \mid s = Ran\ r\}$
Total	$\vdash \forall\ s\bullet\ Total\ s = \{r \mid s = Dom\ r\}$
Functional	$\vdash Functional$ $= \{r \mid \forall\ x\ w\ z\bullet\ (x,\ w) \in r \wedge (x,\ z) \in r \Rightarrow w = z\}$
⊕	$\vdash \forall\ r\ s\bullet\ r \oplus s = (Dom\ s ⩤ r) \cup s$
+	$\vdash \forall\ r\bullet\ r\ ^{+} = \bigcap\ \{q \mid r \subseteq q \wedge q \in Transitive\}$
*	$\vdash \forall\ r$ $\bullet\ r\ ^{*}$ $= \bigcap\ \{q \mid r \subseteq q \wedge q \in Reflexive \wedge q \in Transitive\}$
RelCombine	$\vdash \forall\ f\ g$ $\bullet\ RelCombine\ f\ g$ $= \{(x,\ y,\ z) \mid (x,\ y) \in f \wedge (x,\ z) \in g\}$

9.1.2.8 Theorems

rel_∈_in_clauses

$\vdash \forall\ a\ b\ x\ x1\ y\ z\ r\ r1\ r2\ s\ t\ q\ f$
$\bullet\ (x \mapsto y \in r \Leftrightarrow (x,\ y) \in r)$
$\wedge\ ((x,\ y) \in (a \times b) \Leftrightarrow x \in a \wedge y \in b)$
$\wedge\ (r \in a \leftrightarrow b \Leftrightarrow r \subseteq (a \times b))$
$\wedge\ (x \in Dom\ r \Leftrightarrow (\exists\ y\bullet\ (x,\ y) \in r))$
$\wedge\ (y \in Ran\ r \Leftrightarrow (\exists\ x\bullet\ (x,\ y) \in r))$
$\wedge\ ((x,\ x1) \in Id\ a \Leftrightarrow x = x1 \wedge x \in a)$
$\wedge\ ((x,\ y) \in Graph\ f \Leftrightarrow y = f\ x)$
$\wedge\ ((x,\ z) \in r\ ⨾\ s$
$\Leftrightarrow (\exists\ y\bullet\ (x,\ y) \in r \wedge (y,\ z) \in s))$
$\wedge\ ((x,\ z) \in s\ o\ r \Leftrightarrow (x,\ z) \in r\ ⨾\ s)$
$\wedge\ ((x,\ y) \in a \lhd r \Leftrightarrow x \in a \wedge (x,\ y) \in r)$
$\wedge\ ((x,\ y) \in r \rhd b \Leftrightarrow y \in b \wedge (x,\ y) \in r)$
$\wedge\ ((x,\ y) \in a ⩤ r \Leftrightarrow \neg\ x \in a \wedge (x,\ y) \in r)$
$\wedge\ ((x,\ y) \in r ⩥ b \Leftrightarrow \neg\ y \in b \wedge (x,\ y) \in r)$
$\wedge\ ((y,\ x) \in r\ {}^{\sim} \Leftrightarrow (x,\ y) \in r)$
$\wedge\ (y \in r\ Image\ a \Leftrightarrow (\exists\ x\bullet\ x \in a \wedge (x,\ y) \in r))$
$\wedge\ (q \in Reflexive \Leftrightarrow (\forall\ x\bullet\ (x,\ x) \in q))$
$\wedge\ (q \in Symmetric$
$\Leftrightarrow (\forall\ x1\ x2\bullet\ (x1,\ x2) \in q \Rightarrow (x2,\ x1) \in q))$
$\wedge\ (q \in Transitive$
$\Leftrightarrow (\forall\ x1\ x2\ x3$
$\bullet\ (x1,\ x2) \in q \wedge (x2,\ x3) \in q \Rightarrow (x1,\ x3) \in q))$
$\wedge\ (r \in Injective$
$\Leftrightarrow (\forall\ x1\ x2\ y$
$\bullet\ (x1,\ y) \in r \wedge (x2,\ y) \in r \Rightarrow x1 = x2))$
$\wedge\ (r \in Surjective\ b$
$\Leftrightarrow (\forall\ y\bullet\ y \in b \Leftrightarrow (\exists\ x\bullet\ (x,\ y) \in r)))$
$\wedge\ (r \in Total\ a$
$\Leftrightarrow (\forall\ x\bullet\ x \in a \Leftrightarrow (\exists\ y\bullet\ (x,\ y) \in r)))$
$\wedge\ (r \in Functional$
$\Leftrightarrow (\forall\ x\ y1\ y2$
$\bullet\ (x,\ y1) \in r \wedge (x,\ y2) \in r \Rightarrow y1 = y2))$
$\wedge\ ((x,\ y) \in r1 \oplus r2$
$\Leftrightarrow (x,\ y) \in (Dom\ r2 ⩤ r1) \cup r2)$
$\wedge\ ((x,\ y,\ z) \in RelCombine\ r\ t$
$\Leftrightarrow (x,\ y) \in r \wedge (x,\ z) \in t)$

bin_rel_ext_clauses

$\vdash \forall\ r1\ r2$
$\bullet\ (r1 \subset r2$
$\Leftrightarrow (\forall\ x\ y\bullet\ (x,\ y) \in r1 \Rightarrow (x,\ y) \in r2)$
$\wedge\ (\exists\ x\ y\bullet\ \neg\ (x,\ y) \in r1 \wedge (x,\ y) \in r2))$
$\wedge\ (r1 \subseteq r2 \Leftrightarrow (\forall\ x\ y\bullet\ (x,\ y) \in r1 \Rightarrow (x,\ y) \in r2))$
$\wedge\ (r1 = r2 \Leftrightarrow (\forall\ x\ y\bullet\ (x,\ y) \in r1 \Leftrightarrow (x,\ y) \in r2))$

inv_rel_thm $\vdash \forall\ f\ a\ b$
$\bullet\ (f\ {}^{\sim} \in Functional \Leftrightarrow f \in Injective)$
$\wedge\ (f\ {}^{\sim} \in Injective \Leftrightarrow f \in Functional)$
$\wedge\ (f\ {}^{\sim} \in Surjective\ a \Leftrightarrow f \in Total\ a)$
$\wedge\ (f\ {}^{\sim} \in Total\ b \Leftrightarrow f \in Surjective\ b)$

bin_rel_∅_universe_thm

$\vdash \forall\ f\ g\ r0\ r1\ a\ b$
$\bullet\ (Dom\ r0 = \{\} \Leftrightarrow r0 = \{\})$
$\land\ (Ran\ r0 = \{\} \Leftrightarrow r0 = \{\})$
$\land\ Dom\ \{\} = \{\}$
$\land\ Ran\ \{\} = \{\}$
$\land\ Dom\ Universe = Universe$
$\land\ Ran\ Universe = Universe$
$\land\ (Id\ r0 = \{\} \Leftrightarrow r0 = \{\})$
$\land\ Id\ \{\} = \{\}$
$\land\ (r0\ {}^{\sim} = \{\} \Leftrightarrow r0 = \{\})$
$\land\ \{\}\ {}^{\sim} = \{\}$
$\land\ Universe\ {}^{\sim} = Universe$
$\land\ r0 \fatsemi \{\} = \{\}$
$\land\ \{\} \fatsemi r0 = \{\}$
$\land\ (\{\} = r0 \fatsemi r1 \Leftrightarrow Ran\ r0 \cap Dom\ r1 = \{\})$
$\land\ (r0 \fatsemi r1 = \{\} \Leftrightarrow Ran\ r0 \cap Dom\ r1 = \{\})$
$\land\ RelCombine\ r0\ \{\} = \{\}$
$\land\ RelCombine\ \{\}\ r0 = \{\}$
$\land\ r0\ Image\ \{\} = \{\}$
$\land\ \{\}\ Image\ a = \{\}$
$\land\ f \oplus \{\} = f$
$\land\ \{\} \oplus f = f$
$\land\ (f \oplus g = \{\} \Leftrightarrow f = \{\} \land g = \{\})$
$\land\ (f = \{\} \land g = \{\} \Rightarrow f \oplus g = \{\})$
$\land\ (f \oplus g = \{\} \Leftrightarrow f = \{\} \land g = \{\})$
$\land\ f \oplus Universe = Universe$
$\land\ Universe \mathbin{⩤} r0 = \{\}$
$\land\ a \mathbin{⩤} \{\} = \{\}$
$\land\ \{\} \mathbin{⩤} r0 = r0$
$\land\ Universe \lhd r0 = r0$
$\land\ a \lhd \{\} = \{\}$
$\land\ \{\} \lhd r0 = \{\}$
$\land\ r0 \mathbin{⩥} Universe = \{\}$
$\land\ \{\} \mathbin{⩥} b = \{\}$
$\land\ r0 \mathbin{⩥} \{\} = r0$
$\land\ r0 \rhd Universe = r0$
$\land\ \{\} \rhd b = \{\}$
$\land\ r0 \rhd \{\} = \{\}$

bin_rel_insert_thm

$\vdash \forall\ a\ b\ r\ x\ xy\ y$
$\bullet\ Dom\ (Insert\ (x,\ y)\ r) = Insert\ x\ (Dom\ r)$
$\land\ Dom\ (Insert\ xy\ r) = Insert\ (Fst\ xy)\ (Dom\ r)$
$\land\ Ran\ (Insert\ (x,\ y)\ r) = Insert\ y\ (Ran\ r)$
$\land\ Ran\ (Insert\ xy\ r) = Insert\ (Snd\ xy)\ (Ran\ r)$
$\land\ Id\ (Insert\ x\ a) = Insert\ (x,\ x)\ (Id\ a)$
$\land\ Insert\ (x,\ y)\ r\ Image\ a$
$= r\ Image\ a \cup (if\ x \in a\ then\ \{y\}\ else\ \{\})$
$\land\ Insert\ (x,\ y)\ r \mathbin{⩥} b$
$= (r \mathbin{⩥} b) \cup (if\ \neg\ y \in b\ then\ \{(x,\ y)\}\ else\ \{\})$
$\land\ Insert\ (x,\ y)\ r \rhd b$

$$
\begin{aligned}
&\quad = (r \rhd b) \cup (\textit{if}\ \ y \in b\ \ \textit{then}\ \{(x,\ y)\}\ \textit{else}\ \{\})\\
&\land\ a \unlhd \textit{Insert}\ (x,\ y)\ r\\
&\quad = (a \unlhd r) \cup (\textit{if}\ \ \neg\ x \in a\ \ \textit{then}\ \{(x,\ y)\}\ \textit{else}\ \{\})\\
&\land\ a \lhd \textit{Insert}\ (x,\ y)\ r\\
&\quad = (a \lhd r) \cup (\textit{if}\ \ x \in a\ \ \textit{then}\ \{(x,\ y)\}\ \textit{else}\ \{\})\\
&\land\ \textit{Insert}\ (x,\ y)\ r\ {}^{\sim} = \textit{Insert}\ (y,\ x)\ (r\ {}^{\sim})
\end{aligned}
$$

9.1.3 The Theory char

9.1.3.1 Parents

list

9.1.3.2 Children

basic_hol

9.1.3.3 Constants

IsCharRep	$\mathbb{N} \to BOOL$
RepChar	$CHAR \to \mathbb{N}$
AbsChar	$\mathbb{N} \to CHAR$

9.1.3.4 Types

CHAR

9.1.3.5 Type Abbreviations

STRING *CHAR LIST*

9.1.3.6 Definitions

IsCharRep
is_char_rep_def
$\vdash \forall\ x \bullet\ IsCharRep\ x \Leftrightarrow x < 256$
CHAR
char_def $\vdash \exists\ f \bullet\ TypeDefn\ IsCharRep\ f$
AbsChar
RepChar
abs_char_rep_char_def
$\vdash (\forall\ a \bullet\ AbsChar\ (RepChar\ a) = a)$
$\quad \wedge (\forall\ r \bullet\ IsCharRep\ r \Leftrightarrow RepChar\ (AbsChar\ r) = r)$

9.1.4 The Theory combin

9.1.4.1 Parents

basic_hol

9.1.4.2 Children

sum

9.1.4.3 Constants

$o	$('b \to 'c) \to ('a \to 'b) \to 'a \to 'c$
CombS	$('a \to 'b \to 'c) \to ('a \to 'b) \to 'a \to 'c$
CombK	$'a \to 'b \to 'a$
CombI	$'a \to 'a$

9.1.4.4 Fixity

Right Infix 300:

o

9.1.4.5 Definitions

o	
o_def	$\vdash \forall\ f\ g\ x \bullet\ (f\ o\ g)\ x = f\ (g\ x)$
CombS	
comb_s_def	$\vdash \forall\ f\ g\ x \bullet\ CombS\ f\ g\ x = f\ x\ (g\ x)$
CombK	
comb_k_def	$\vdash \forall\ x\ y \bullet\ CombK\ x\ y = x$
CombI	
comb_i_def	$\vdash \forall\ x \bullet\ CombI\ x = x$

9.1.4.6 Theorems

s_k_thm	$\vdash \forall\ x \bullet\ CombS\ CombK\ x = CombI$
o_assoc_thm	$\vdash \forall\ f\ g\ h \bullet\ f\ o\ g\ o\ h = (f\ o\ g)\ o\ h$
o_i_thm	$\vdash \forall\ f \bullet\ CombI\ o\ f = f \wedge f\ o\ CombI = f$

9.1.5 The Theory dyadic

9.1.5.1 Parents

$\mathbb{Z}$

9.1.5.2 Children

$\mathbb{R}$

9.1.5.3 Constants

$^	$\mathbb{N} \to \mathbb{N} \to \mathbb{N}$
$dy_times	$\mathbb{N} \times \mathbb{Z} \to \mathbb{N} \times \mathbb{Z} \to \mathbb{N} \times \mathbb{Z}$
dy_one	$\mathbb{N} \times \mathbb{Z}$
$dy_exp	$\mathbb{N} \times \mathbb{Z} \to \mathbb{N} \to \mathbb{N} \times \mathbb{Z}$
$dy_less	$\mathbb{N} \times \mathbb{Z} \to \mathbb{N} \times \mathbb{Z} \to BOOL$

9.1.5.4 Type Abbreviations

DYADIC	$\mathbb{N} \times \mathbb{Z}$

9.1.5.5 Fixity

Right Infix 210:

dy_less

Right Infix 310:

dy_exp **dy_times**

Right Infix 320:

^

9.1.5.6 Definitions

^ $\vdash (\forall\ i\bullet\ i\ \hat{}\ 0 = 1) \land (\forall\ i\ m\bullet\ i\ \hat{}\ (m + 1) = i * i\ \hat{}\ m)$

dy_times $\vdash \forall\ (m, i)\ (n, j)$
$\bullet\ (m, i)\ dy\_times\ (n, j) = (2 * m * n + m + n, i + j)$

dy_one $\vdash dy\_one = (0, \mathbb{N}\mathbb{Z}\ 0)$

dy_exp $\vdash (\forall\ x\bullet\ x\ dy\_exp\ 0 = dy\_one)$
$\land (\forall\ x\ m\bullet\ x\ dy\_exp\ (m + 1) = x\ dy\_times\ x\ dy\_exp\ m)$

dy_less $\vdash \forall\ (m, i)\ (n, j)$
$\bullet\ (m, i)\ dy\_less\ (n, j)$
$\Leftrightarrow (\exists\ a\ b$
$\bullet\ \mathbb{N}\mathbb{Z}\ a + i = \mathbb{N}\mathbb{Z}\ b + j$
$\land\ 2\ \hat{}\ a * (2 * m + 1) < 2\ \hat{}\ b * (2 * n + 1))$

9.1.5.7 Theorems

dy_less_irrefl_thm
$\vdash \forall\ x \bullet\ \neg\ x\ dy\_less\ x$

dy_less_antisym_thm
$\vdash \forall\ x\ y \bullet\ \neg\ (x\ dy\_less\ y \wedge y\ dy\_less\ x)$

dy_less_trans_thm
$\vdash \forall\ x\ y\ z \bullet\ x\ dy\_less\ y \wedge y\ dy\_less\ z \Rightarrow x\ dy\_less\ z$

dy_less_trich_thm
$\vdash \forall\ x\ y \bullet\ \neg\ x = y \Rightarrow x\ dy\_less\ y \vee y\ dy\_less\ x$

dy_times_comm_thm
$\vdash \forall\ x\ y \bullet\ x\ dy\_times\ y = y\ dy\_times\ x$

dy_times_assoc_thm
$\vdash \forall\ x\ y\ z$
$\quad \bullet\ (x\ dy\_times\ y)\ dy\_times\ z = x\ dy\_times\ y\ dy\_times\ z$

dy_times_order_thm
$\vdash \forall\ u\ x\ y$
$\quad \bullet\ x\ dy\_times\ u = u\ dy\_times\ x$
$\qquad \wedge (u\ dy\_times\ x)\ dy\_times\ y$
$\qquad\quad = u\ dy\_times\ x\ dy\_times\ y$
$\qquad \wedge x\ dy\_times\ u\ dy\_times\ y$
$\qquad\quad = u\ dy\_times\ x\ dy\_times\ y$

dy_times_unit_thm
$\vdash \forall\ x \bullet\ x\ dy\_times\ dy\_one = x$

dy_times_unit_clauses
$\vdash \forall\ x \bullet\ x\ dy\_times\ dy\_one = x \wedge dy\_one\ dy\_times\ x = x$

dy_exp_clauses
$\vdash \forall\ x\ m\ n$
$\quad \bullet\ x\ dy\_exp\ (m + n) = (x\ dy\_exp\ m)\ dy\_times\ x\ dy\_exp\ n$
$\qquad \wedge x\ dy\_exp\ 0 = dy\_one$
$\qquad \wedge x\ dy\_exp\ 1 = x$
$\qquad \wedge x\ dy\_exp\ 2 = x\ dy\_times\ x$
$\qquad \wedge x\ dy\_exp\ 3 = x\ dy\_times\ x\ dy\_times\ x$

dy_times_mono_thm
$\vdash \forall\ x\ y\ z$
$\quad \bullet\ y\ dy\_less\ z \Rightarrow x\ dy\_times\ y\ dy\_less\ x\ dy\_times\ z$

dy_times_mono_thm1
$\vdash \forall\ x\ y\ z$
$\quad \bullet\ y\ dy\_less\ z \Rightarrow y\ dy\_times\ x\ dy\_less\ z\ dy\_times\ x$

dy_times_mono_thm2
$\vdash \forall\ x\ y\ v\ w$
$\quad \bullet\ x\ dy\_less\ y \wedge v\ dy\_less\ w$
$\qquad \Rightarrow x\ dy\_times\ v\ dy\_less\ y\ dy\_times\ w$

dy_times_mono_⇔_thm
$\vdash \forall\ x\ y\ z$
$\quad \bullet\ x\ dy\_times\ y\ dy\_less\ x\ dy\_times\ z \Leftrightarrow y\ dy\_less\ z$

dy_arch_thm $\vdash \forall\ x\ y \bullet\ dy\_one\ dy\_less\ x \Rightarrow (\exists\ t \bullet\ y\ dy\_less\ x\ dy\_exp\ t)$

dy_balance_thm1
$\vdash \forall\ x \bullet\ \exists\ y \bullet\ dy\_one\ dy\_less\ x\ dy\_times\ y$

dy_balance_thm2
$\vdash \forall\ x \bullet\ \exists\ y \bullet\ x\ dy\_times\ y\ dy\_less\ dy\_one$

dy_right_dense_thm

$\vdash \forall\ x\ y$
$\quad\bullet\ x\ dy\_less\ y$
$\qquad\Rightarrow (\exists\ z$
$\qquad\bullet\ x\ dy\_less\ x\ dy\_times\ z$
$\qquad\quad\wedge\ x\ dy\_times\ z\ dy\_less\ y)$

dy_less_dense_thm

$\vdash \forall\ x\ y \bullet\ x\ dy\_less\ y \Rightarrow (\exists\ z \bullet\ x\ dy\_less\ z \wedge z\ dy\_less\ y)$

dy_left_dense_thm

$\vdash \forall\ x\ y$
$\quad\bullet\ x\ dy\_less\ y$
$\qquad\Rightarrow (\exists\ z$
$\qquad\bullet\ x\ dy\_less\ y\ dy\_times\ z$
$\qquad\quad\wedge\ y\ dy\_times\ z\ dy\_less\ y)$

9.1.6 The Theory fin_set

9.1.6.1 Parents

seq

9.1.6.2 Constants

$\mathbb{N}$	$\mathbb{N}\ SET$
Finite	$'a\ SET\ SET$
$\mathbb{F}$	$'a\ SET \to 'a\ SET\ SET$
$\mathbb{F}_1$	$'a\ SET \to 'a\ SET\ SET$
Min	$\mathbb{N}\ SET \to \mathbb{N}$
Max	$\mathbb{N}\ SET \to \mathbb{N}$
Size	$'a\ SET \to \mathbb{N}$
Iter	$\mathbb{N} \to ('a \times 'a)\ SET \to ('a \times 'a)\ SET$
$\$\twoheadrightarrow\!\!\!\!\!\!+$	$'a\ SET \to 'b\ SET \to ('a \times 'b)\ SET\ SET$
$\$\rightarrowtail\!\!\!\!\!\!+\!\!\!\!+$	$'a\ SET \to 'b\ SET \to ('a \times 'b)\ SET\ SET$

9.1.6.3 Aliases

#	$Size : 'a\ SET \to \mathbb{N}$

9.1.6.4 Fixity

Right Infix 240:

$\rightarrowtail\!\!\!\!\!\!+\!\!\!\!+ \qquad \twoheadrightarrow\!\!\!\!\!\!+$

9.1.6.5 Definitions

$\mathbb{N}$ $\quad \vdash \mathbb{N} = Universe$

Finite $\quad \vdash Finite = \bigcap\ \{u | \{\} \in u \land (\forall\ a\ x \bullet\ a \in u \Rightarrow \{x\} \cup a \in u)\}$

$\mathbb{F}$ $\quad \vdash \forall\ x \bullet\ \mathbb{F}\ x = \mathbb{P}\ x \cap Finite$

$\mathbb{F}_1$ $\quad \vdash \forall\ x \bullet\ \mathbb{F}_1\ x = \mathbb{F}\ x \setminus \{\{\}\}$

Min
$$\begin{array}{l}
\vdash ConstSpec \\
\quad (\lambda\ Min' \\
\quad\quad \bullet \forall\ m\ a \\
\quad\quad\quad \bullet\ m \in a \land (\forall\ i \bullet\ i \in a \Rightarrow m \le i) \Rightarrow Min'\ a = m) \\
\quad Min
\end{array}$$

Max
$$\begin{array}{l}
\vdash ConstSpec \\
\quad (\lambda\ Max' \\
\quad\quad \bullet \forall\ m\ a \\
\quad\quad\quad \bullet\ m \in a \land (\forall\ i \bullet\ i \in a \Rightarrow i \le m) \Rightarrow Max'\ a = m) \\
\quad Max
\end{array}$$

Size
$$\begin{array}{l}
\vdash \forall\ a \\
\quad \bullet\ \#\ a = Min\ \{i | \exists\ list \bullet\ \#\ list = i \land Elems\ list = a\}
\end{array}$$

Iter
$$\begin{array}{l}
\vdash \forall\ r\ n \\
\quad \bullet\ Iter\ 0\ r = Id\ Universe \\
\quad\quad \land\ Iter\ (n + 1)\ r = r \fatsemi Iter\ n\ r
\end{array}$$

$\twoheadrightarrow\!\!\!\!\!\!+$ $\quad \vdash \forall\ a\ b \bullet\ a \twoheadrightarrow\!\!\!\!\!\!+\ b = (a \leftrightarrow\!\!\!\!\!\!+\ b) \cap Finite$

$\rightarrowtail\!\!\!\!\!\!+\!\!\!\!+$ $\quad \vdash \forall\ a\ b \bullet\ a \rightarrowtail\!\!\!\!\!\!+\!\!\!\!+\ b = (a \twoheadrightarrow\!\!\!\!\!\!+\ b) \cap (a \rightarrowtail\!\!\!\!\!\!+\ b)$

9.1.6.6 Theorems

Min_consistent

$\vdash$ *Consistent* $(\lambda\ Min' \bullet \forall\ m\ a \bullet m \in a \land (\forall\ i \bullet i \in a \Rightarrow m \leq i) \Rightarrow Min'\ a = m)$

Max_consistent

$\vdash$ *Consistent* $(\lambda\ Max' \bullet \forall\ m\ a \bullet m \in a \land (\forall\ i \bullet i \in a \Rightarrow i \leq m) \Rightarrow Max'\ a = m)$

finite_induction_thm

$\vdash \forall\ p \bullet p\ \{\} \land (\forall\ a\ x \bullet a \in Finite \land p\ a \land \neg\ x \in a \Rightarrow p\ (\{x\} \cup a)) \Rightarrow (\forall\ a \bullet a \in Finite \Rightarrow p\ a)$

empty_finite_thm

$\vdash \{\} \in Finite$

singleton_∪_finite_thm

$\vdash \forall\ a\ x \bullet a \in Finite \Rightarrow \{x\} \cup a \in Finite$

⊆_finite_thm $\vdash \forall\ a\ b \bullet a \in Finite \land b \subseteq a \Rightarrow b \in Finite$

∪_finite_thm $\vdash \forall\ a\ b \bullet a \cup b \in Finite \Leftrightarrow a \in Finite \land b \in Finite$

∩_finite_thm $\vdash \forall\ a\ b \bullet a \in Finite \lor b \in Finite \Rightarrow a \cap b \in Finite$

finite_distinct_elems_thm

$\vdash \forall\ a \bullet a \in Finite \Rightarrow (\exists\ list \bullet list \in Distinct \land Elems\ list = a)$

length_↾_≤_thm

$\vdash \forall\ list\ a \bullet \#\ (list \restriction a) \leq \#\ list$

length_↾_less_thm

$\vdash \forall\ list\ a \bullet \neg\ Elems\ list \setminus a = \{\} \Rightarrow \#\ (list \restriction a) < \#\ list$

elems_↾_thm $\vdash \forall\ list\ a \bullet Elems\ (list \restriction a) = Elems\ list \cap a$

distinct_length_≤_thm

$\vdash \forall\ list1\ list2 \bullet list1 \in Distinct \land Elems\ list1 = Elems\ list2 \Rightarrow \#\ list1 \leq \#\ list2$

distinct_size_length_thm

$\vdash \forall\ list\ a \bullet list \in Distinct \land Elems\ list = a \Rightarrow \#\ a = \#\ list$

size_empty_thm

$\vdash \#\ \{\} = 0$

size_singleton_∪_thm

$\vdash \forall\ x\ a \bullet a \in Finite \land \neg\ x \in a \Rightarrow \#\ (\{x\} \cup a) = \#\ a + 1$

size_singleton_thm

$\vdash \forall\ x \bullet \#\ \{x\} = 1$

size_∪_thm $\vdash \forall\ a\ b \bullet a \in Finite \land b \in Finite \Rightarrow \#\ (a \cup b) + \#\ (a \cap b) = \#\ a + \#\ b$

size_0_thm $\vdash \forall\ a \bullet a \in Finite \Rightarrow (\#\ a = 0 \Leftrightarrow a = \{\})$

size_1_thm $\vdash \forall\ a\bullet\ a \in \mathit{Finite} \Rightarrow (\#\ a = \mathit{1} \Leftrightarrow (\exists\ x\bullet\ a = \{x\}))$
$\bigcup$_finite_thm $\vdash \forall\ u\bullet\ u \in \mathit{Finite} \land u \subseteq \mathit{Finite} \Rightarrow \bigcup\ u \in \mathit{Finite}$
pigeon_hole_thm
$\vdash \forall\ u$
$\bullet\ u \in \mathit{Finite} \land u \subseteq \mathit{Finite} \land \#\ u < \#\ (\bigcup\ u)$
$\Rightarrow (\exists\ a\bullet\ a \in u \land \#\ a > \mathit{1})$
$\subseteq$_size_$\leq$_thm $\vdash \forall\ a\ b\bullet\ a \in \mathit{Finite} \land b \subseteq a \Rightarrow \#\ b \leq \#\ a$
$\subseteq$_size_less_thm
$\vdash \forall\ a\ b\bullet\ a \in \mathit{Finite} \land b \subseteq a \land \neg\ b = a \Rightarrow \#\ b < \#\ a$
min_$\in$_thm $\vdash \forall\ n\ a\bullet\ n \in a \Rightarrow \mathit{Min}\ a \in a$
min_$\leq$_thm $\vdash \forall\ n\ a\bullet\ n \in a \Rightarrow \mathit{Min}\ a \leq n$
max_$\in$_thm $\vdash \forall\ m\ n\ a\bullet\ (\forall\ i\bullet\ i \in a \Rightarrow i \leq m) \land n \in a \Rightarrow \mathit{Max}\ a \in a$
$\leq$_max_thm $\vdash \forall\ m\ n\ a\bullet\ (\forall\ i\bullet\ i \in a \Rightarrow i \leq m) \land n \in a \Rightarrow n \leq \mathit{Max}\ a$
finite_$\subseteq$_well_founded_thm
$\vdash \forall\ p\ a$
$\bullet\ a \in \mathit{Finite} \land p\ a$
$\Rightarrow (\exists\ b\bullet\ b \subseteq a \land p\ b \land (\forall\ c\bullet\ c \subseteq b \land p\ c \Rightarrow c = b))$

9.1.7 The Theory fun_rel

9.1.7.1 Parents

bin_rel

9.1.7.2 Children

fun_rel_thms *seq*

9.1.7.3 Constants

$\$\nrightarrow$	$'a\ SET \to 'b\ SET \to ('a \times 'b)\ SET\ SET$
$\$\rightarrow$	$'a\ SET \to 'b\ SET \to ('a \times 'b)\ SET\ SET$
$\$\rightarrowtail\!\!\nrightarrow$	$'a\ SET \to 'b\ SET \to ('a \times 'b)\ SET\ SET$
$\$\rightarrowtail$	$'a\ SET \to 'b\ SET \to ('a \times 'b)\ SET\ SET$
$\$\nrightarrow\!\!\!\twoheadrightarrow$	$'a\ SET \to 'b\ SET \to ('a \times 'b)\ SET\ SET$
$\$\twoheadrightarrow$	$'a\ SET \to 'b\ SET \to ('a \times 'b)\ SET\ SET$
$\$\rightarrowtail\!\!\!\twoheadrightarrow$	$'a\ SET \to 'b\ SET \to ('a \times 'b)\ SET\ SET$
$\$$**At**	$('a \times 'b)\ SET \to 'a \to 'b$

9.1.7.4 Aliases

@	$\$At : ('a \times 'b)\ SET \to 'a \to 'b$

9.1.7.5 Fixity

Left Infix 600:

At

Right Infix 240:

$\rightarrowtail\!\!\!\twoheadrightarrow \quad \nrightarrow\!\!\!\twoheadrightarrow \quad \nrightarrow \quad \rightarrowtail \quad \twoheadrightarrow \quad \rightarrowtail\!\!\nrightarrow$

9.1.7.6 Definitions

$\nrightarrow$ $\quad \vdash \forall\ a\ b \bullet\ a \nrightarrow b = (a \leftrightarrow b) \cap Functional$

$\rightarrow$ $\quad \vdash \forall\ a\ b \bullet\ (a \rightarrow b) = (a \leftrightarrow b) \cap Functional \cap Total\ a$

$\rightarrowtail\!\!\nrightarrow$ $\quad \vdash \forall\ a\ b \bullet\ a \rightarrowtail\!\!\nrightarrow b = (a \leftrightarrow b) \cap Functional \cap Injective$

$\rightarrowtail$ $\quad \vdash \forall\ a\ b$
$\quad\quad \bullet\ a \rightarrowtail b = (a \leftrightarrow b) \cap Functional \cap Injective \cap Total\ a$

$\nrightarrow\!\!\!\twoheadrightarrow$ $\quad \vdash \forall\ a\ b \bullet\ a \nrightarrow\!\!\!\twoheadrightarrow b = (a \leftrightarrow b) \cap Functional \cap Surjective\ b$

$\twoheadrightarrow$ $\quad \vdash \forall\ a\ b$
$\quad\quad \bullet\ a \twoheadrightarrow b$
$\quad\quad\quad = (a \leftrightarrow b) \cap Functional \cap Surjective\ b \cap Total\ a$

$\rightarrowtail\!\!\!\twoheadrightarrow$ $\quad \vdash \forall\ a\ b$
$\quad\quad \bullet\ a \rightarrowtail\!\!\!\twoheadrightarrow b$
$\quad\quad\quad = (a \leftrightarrow b)$
$\quad\quad\quad\quad \cap\ Functional$
$\quad\quad\quad\quad \cap\ Injective$
$\quad\quad\quad\quad \cap\ Surjective\ b$
$\quad\quad\quad\quad \cap\ Total\ a$

At $\vdash ConstSpec$
$(\lambda\ At'$
$\bullet \forall\ f\ x\ y$
$\bullet (x,\ y) \in f \land (\forall\ z \bullet (x,\ z) \in f \Rightarrow z = y)$
$\Rightarrow At'\ f\ x = y)$
\$@

9.1.7.7 Theorems

At_consistent
$\vdash Consistent$
$(\lambda\ At'$
$\bullet \forall\ f\ x\ y$
$\bullet (x,\ y) \in f \land (\forall\ z \bullet (x,\ z) \in f \Rightarrow z = y)$
$\Rightarrow At'\ f\ x = y)$

graph_at_thm $\vdash \forall\ f\ x \bullet Graph\ f\ @\ x = f\ x$

inv_rel_∈_arrow_thm
$\vdash \forall\ f\ a\ b$
$\bullet (f\ ^{\sim} \in b \leftrightarrow a \Leftrightarrow f \in a \leftrightarrow b)$
$\land (f\ ^{\sim} \in b \nrightarrow a \Leftrightarrow f \in (a \leftrightarrow b) \cap Injective)$
$\land (f\ ^{\sim} \in (b \rightarrow a)$
$\Leftrightarrow f \in (a \leftrightarrow b) \cap Injective \cap Surjective\ b)$
$\land (f\ ^{\sim} \in b \rightarrowtail\!\!\!\!\nrightarrow a$
$\Leftrightarrow f \in (a \leftrightarrow b) \cap Injective \cap Functional)$
$\land (f\ ^{\sim} \in b \twoheadrightarrow\!\!\!\!\!\!\nrightarrow a$
$\Leftrightarrow f \in (a \leftrightarrow b) \cap Injective \cap Total\ a)$
$\land (f\ ^{\sim} \in b \twoheadrightarrow a$
$\Leftrightarrow f$
$\in (a \leftrightarrow b)$
$\cap Injective$
$\cap Total\ a$
$\cap Surjective\ b)$
$\land (f\ ^{\sim} \in b \rightarrowtail\!\!\!\!\twoheadrightarrow a$
$\Leftrightarrow f$
$\in (a \leftrightarrow b)$
$\cap Injective$
$\cap Functional$
$\cap Surjective\ b$
$\cap Total\ a)$
$\land (f\ ^{\sim} \in b \rightarrowtail a$
$\Leftrightarrow f$
$\in (a \leftrightarrow b)$
$\cap Injective$
$\cap Functional$
$\cap Surjective\ b)$

at_at_eq_thm $\vdash \forall\ f\ X\ Y\ x\ y$
$\bullet f \in X \nrightarrow Y \land x \in Dom\ f \land y \in Dom\ f$
$\Rightarrow (f\ @\ x = f\ @\ y$
$\Leftrightarrow (\exists\ z \bullet (x,\ z) \in f \land (y,\ z) \in f))$

9.1.8 The Theory fun_rel_thms

9.1.8.1 Parents

set_thms *fun_rel*

9.1.8.2 Constants

ReflexiveIn $'a\ SET \rightarrow ('a \times 'a)\ SET\ SET$

Antisymmetric

$('a \times 'a)\ SET\ SET$

Chains $('a \times 'a)\ SET \rightarrow 'a\ SET\ SET$

UpperBounds $('a \times 'a)\ SET \rightarrow 'a\ SET \rightarrow 'a\ SET$

MaximalElements

$('a \times 'a)\ SET \rightarrow 'a\ SET \rightarrow 'a\ SET$

9.1.8.3 Definitions

ReflexiveIn $\vdash \forall\ a\bullet\ ReflexiveIn\ a = \{r|\forall\ x\bullet\ x \in a \Rightarrow (x,\ x) \in r\}$

Antisymmetric

$\vdash Antisymmetric$
$= \{r|\forall\ x\ y\bullet\ (x,\ y) \in r \wedge (y,\ x) \in r \Rightarrow x = y\}$

Chains $\vdash \forall\ r$
$\bullet\ Chains\ r$
$= \{a$
$|\forall\ x\ y\bullet\ x \in a \wedge y \in a \Rightarrow (x,\ y) \in r \vee (y,\ x) \in r\}$

UpperBounds $\vdash \forall\ r\ a\bullet\ UpperBounds\ r\ a = \{x|\forall\ y\bullet\ y \in a \Rightarrow (y,\ x) \in r\}$

MaximalElements

$\vdash \forall\ r\ a$
$\bullet\ MaximalElements\ r\ a$
$= \{x|x \in a \wedge (\forall\ y\bullet\ y \in a \wedge (x,\ y) \in r \Rightarrow y = x)\}$

9.1.8.4 Theorems

zorn_thm $\vdash \forall\ r\ a$
$\bullet\ r \in ReflexiveIn\ a$
$\wedge\ r \in Transitive$
$\wedge\ r \in Antisymmetric$
$\wedge\ (\forall\ c$
$\bullet\ c \subseteq a \wedge c \in Chains\ r$
$\Rightarrow (\exists\ x\bullet\ x \in a \wedge x \in UpperBounds\ r\ c))$
$\Rightarrow (\exists\ x\bullet\ x \in MaximalElements\ r\ a)$

comparability_thm

$\vdash \forall\ a\ b\bullet\ (\exists\ f\bullet\ f \in a \rightarrowtail b) \vee (\exists\ g\bullet\ g \in b \rightarrowtail a)$

schroeder_bernstein_thm1

$\vdash \forall\ a\ b\ c\ f$
$\bullet\ c \subseteq b \wedge b \subseteq a \wedge f \in a \rightarrowtail c \Rightarrow (\exists\ h\bullet\ h \in a \twoheadrightarrow b)$

schroeder_bernstein_thm

$\vdash \forall\ a\ b\ f\ g\bullet\ f \in a \rightarrowtail b \wedge g \in b \rightarrowtail a \Rightarrow (\exists\ h\bullet\ h \in a \twoheadrightarrow b)$

9.1.9 The Theory hol

9.1.9.1 Parents

sets *sum* *one*

9.1.9.2 Children

bin_rel

9.1.10 The Theory init

9.1.10.1 Parents

log

9.1.10.2 Children

misc

9.1.10.3 Axioms

bool_cases_axiom
$\vdash \forall\ b\bullet\ (b \Leftrightarrow T) \lor (b \Leftrightarrow F)$

$\Rightarrow$_antisym_axiom
$\vdash \forall\ b1\ \ b2\bullet\ (b1 \Rightarrow b2) \Rightarrow (b2 \Rightarrow b1) \Rightarrow (b1 \Leftrightarrow b2)$

η_axiom $\vdash \forall\ f\bullet\ (\lambda\ x\bullet\ f\ x) = f$

ϵ_axiom $\vdash \forall\ P\ x\bullet\ P\ x \Rightarrow P\ (\$\epsilon\ P)$

infinity_axiom
$\vdash \exists\ f\bullet\ OneOne\ f \land \neg\ Onto\ f$

9.1.11 The Theory list

9.1.11.1 Parents

$\mathbb{N}$

9.1.11.2 Children

char

9.1.11.3 Constants

IsListRep	$(\mathbb{N} \to {'a}) \times \mathbb{N} \to BOOL$
Cons	${'a} \to {'a}\ LIST \to {'a}\ LIST$
[]	${'a}\ LIST$
Length	${'a}\ LIST \to \mathbb{N}$
Hd	${'a}\ LIST \to {'a}$
Tl	${'a}\ LIST \to {'a}\ LIST$
Append	${'a}\ LIST \to {'a}\ LIST \to {'a}\ LIST$
Rev	${'a}\ LIST \to {'a}\ LIST$
Split	$({'a} \times {'b})\ LIST \to {'a}\ LIST \times {'b}\ LIST$
Combine	${'a}\ LIST \to {'b}\ LIST \to ({'a} \times {'b})\ LIST$
Map	$({'a} \to {'b}) \to {'a}\ LIST \to {'b}\ LIST$
Fold	$({'a} \to {'b} \to {'b}) \to {'a}\ LIST \to {'b} \to {'b}$

9.1.11.4 Aliases

@	$Append : {'a}\ LIST \to {'a}\ LIST \to {'a}\ LIST$

9.1.11.5 Types

'1 **LIST**

9.1.11.6 Fixity

Right Infix 300:

@

9.1.11.7 Definitions

IsListRep
is_list_rep_def

$\vdash \exists\ nil\ cons$
$\quad \bullet\ IsListRep\ nil$
$\qquad \wedge\ (\forall\ x\bullet\ IsListRep\ x \Rightarrow (\forall\ h\bullet\ \neg\ cons\ h\ x = nil))$
$\qquad \wedge\ (\forall\ x\ y$
$\qquad \bullet\ IsListRep\ x \wedge IsListRep\ y$
$\qquad\quad \Rightarrow (\forall\ a\ b$
$\qquad\quad \bullet\ cons\ a\ x = cons\ b\ y \Leftrightarrow a = b \wedge x = y))$
$\qquad \wedge\ (\forall\ x$
$\qquad \bullet\ IsListRep\ x \Rightarrow (\forall\ h\bullet\ IsListRep\ (cons\ h\ x)))$
$\qquad \wedge\ (\forall\ p$
$\qquad \bullet\ p\ nil \wedge (\forall\ m\bullet\ p\ m \Rightarrow (\forall\ h\bullet\ p\ (cons\ h\ m)))$
$\qquad\quad \Rightarrow (\forall\ n\bullet\ IsListRep\ n \Rightarrow p\ n))$

LIST
list_def $\vdash \exists\ f\bullet\ TypeDefn\ IsListRep\ f$

Nil
Cons
nil_cons_def $\vdash (\forall\ x\ list\bullet\ \neg\ Cons\ x\ list = [])$
$\quad \wedge\ (\forall\ x1\ x2\ list1\ list2$
$\quad \bullet\ Cons\ x1\ list1 = Cons\ x2\ list2$
$\qquad \Leftrightarrow x1 = x2 \wedge list1 = list2)$
$\quad \wedge\ (\forall\ p$
$\quad \bullet\ p\ [] \wedge (\forall\ list\bullet\ p\ list \Rightarrow (\forall\ x\bullet\ p\ (Cons\ x\ list)))$
$\qquad \Rightarrow (\forall\ list\bullet\ p\ list))$

Length
length_def $\vdash Length\ [] = 0$
$\quad \wedge\ (\forall\ h\ list$
$\quad \bullet\ Length\ (Cons\ h\ list) = Length\ list + 1)$

Hd
hd_def $\vdash \forall\ h\ list\bullet\ Hd\ (Cons\ h\ list) = h$

Tl
tl_def $\vdash \forall\ h\ list\bullet\ Tl\ (Cons\ h\ list) = list$

Append
append_def $\vdash (\forall\ list\bullet\ []\ @\ list = list)$
$\quad \wedge\ (\forall\ h\ list\ list'$
$\quad \bullet\ Cons\ h\ list\ @\ list' = Cons\ h\ (list\ @\ list'))$

Rev
rev_def $\vdash Rev\ [] = []$
$\quad \wedge\ (\forall\ h\ list\bullet\ Rev\ (Cons\ h\ list) = Rev\ list\ @\ [h])$

Split
split_def $\vdash Split\ [] = ([], [])$
$\quad \wedge\ (\forall\ list\ h1\ h2$
$\quad \bullet\ Split\ (Cons\ (h1, h2)\ list)$
$\qquad = (Cons\ h1\ (Fst\ (Split\ list)),$
$\qquad\quad Cons\ h2\ (Snd\ (Split\ list))))$

Combine
combine_def $\vdash Combine\ []\ [] = []$
$\quad \wedge\ (\forall\ h1\ h2\ list1\ list2$
$\quad \bullet\ Combine\ (Cons\ h1\ list1)\ (Cons\ h2\ list2)$

$= Cons\ (h1,\ h2)\ (Combine\ list1\ list2))$

Map

map_def $\vdash (\forall\ g \bullet\ Map\ g\ []\ =\ [])$
$\land (\forall\ h\ g\ list$
$\bullet\ Map\ g\ (Cons\ h\ list)\ =\ Cons\ (g\ h)\ (Map\ g\ list))$

Fold

fold_def $\vdash (\forall\ g\ x \bullet\ Fold\ g\ []\ x\ =\ x)$
$\land (\forall\ h\ g\ x\ list$
$\bullet\ Fold\ g\ (Cons\ h\ list)\ x\ =\ g\ h\ (Fold\ g\ list\ x))$

9.1.11.8 Theorems

list_induction_thm

$\vdash \forall\ p$
$\bullet\ p\ []\ \land (\forall\ list \bullet\ p\ list\ \Rightarrow (\forall\ x \bullet\ p\ (Cons\ x\ list)))$
$\Rightarrow (\forall\ list \bullet\ p\ list)$

list_prim_rec_thm

$\vdash \forall\ n\ c$
$\bullet\ \exists_1\ f$
$\bullet\ f\ []\ =\ n$
$\land (\forall\ list\ a$
$\bullet\ f\ (Cons\ a\ list)\ =\ c\ a\ (f\ list)\ list)$

list_clauses $\vdash \forall\ x1\ x2\ list1\ list2$
$\bullet\ \neg\ Cons\ x1\ list1\ =\ []$
$\land\ \neg\ []\ =\ Cons\ x1\ list1$
$\land (Cons\ x1\ list1\ =\ Cons\ x2\ list2$
$\Leftrightarrow x1\ =\ x2\ \land\ list1\ =\ list2)$
$\land\ Hd\ (Cons\ x1\ list1)\ =\ x1$
$\land\ Tl\ (Cons\ x1\ list1)\ =\ list1$

list_cases_thm

$\vdash \forall\ list1$
$\bullet\ list1\ =\ []\ \lor (\exists\ x\ list2 \bullet\ list1\ =\ Cons\ x\ list2)$

9.1.12 The Theory log

9.1.12.1 Parents

min

9.1.12.2 Children

init

9.1.12.3 Constants

T	$BOOL$
$\$\forall$	$('a \to BOOL) \to BOOL$
$\$\exists$	$('a \to BOOL) \to BOOL$
F	$BOOL$
$\$\neg$	$BOOL \to BOOL$
$\$\wedge$	$BOOL \to BOOL \to BOOL$
$\$\vee$	$BOOL \to BOOL \to BOOL$
OneOne	$('a \to 'b) \to BOOL$
Onto	$('a \to 'b) \to BOOL$
TypeDefn	$('b \to BOOL) \to ('a \to 'b) \to BOOL$

9.1.12.4 Fixity

Binder:	$\forall$	$\exists$
Right Infix 30:		
	$\vee$	
Right Infix 40:		
	$\wedge$	
Prefix 50:	$\neg$	

9.1.12.5 Terminators

$\vee$ $\wedge$ $\neg$ $\exists$ $\forall$

9.1.12.6 Definitions

T	
t_def	$\vdash T \Leftrightarrow (\lambda\ x \bullet\ x) = (\lambda\ x \bullet\ x)$
$\forall$	
$\forall$_def	$\vdash \$\forall = (\lambda\ P \bullet\ P = (\lambda\ x \bullet\ T))$
$\exists$	
$\exists$_def	$\vdash \$\exists = (\lambda\ P \bullet\ P\ (\$\epsilon\ P))$
F	
f_def	$\vdash F \Leftrightarrow (\forall\ b \bullet\ b)$
$\neg$	
$\neg$_def	$\vdash \$\neg = (\lambda\ b \bullet\ b \Rightarrow F)$
$\wedge$	
$\wedge$_def	$\vdash \$\wedge = (\lambda\ b1\ b2 \bullet\ \forall\ b \bullet\ (b1 \Rightarrow b2 \Rightarrow b) \Rightarrow b)$

$\vee$	
$\vee$_def	$\vdash \$\vee = (\lambda\ b1\ b2 \bullet\ \forall\ b \bullet\ (b1 \Rightarrow b) \Rightarrow (b2 \Rightarrow b) \Rightarrow b)$
OneOne	
one_one_def	$\vdash OneOne = (\lambda\ f \bullet\ \forall\ x1\ x2 \bullet\ f\ x1 = f\ x2 \Rightarrow x1 = x2)$
Onto	
onto_def	$\vdash Onto = (\lambda\ f \bullet\ \forall\ y \bullet\ \exists\ x \bullet\ y = f\ x)$
TypeDefn	
type_defn_def	
	$\vdash TypeDefn$ $= (\lambda\ P\ rep$ $\bullet\ OneOne\ rep \wedge (\forall\ x \bullet\ P\ x \Leftrightarrow (\exists\ y \bullet\ x = rep\ y)))$

9.1.13 The Theory min

9.1.13.1 Children

log

9.1.13.2 Constants

$\$\Rightarrow$	$BOOL \rightarrow BOOL \rightarrow BOOL$
$\$=$	$'a \rightarrow 'a \rightarrow BOOL$
$\$\epsilon$	$('a \rightarrow BOOL) \rightarrow 'a$

9.1.13.3 Types

$'1 \rightarrow '2$
BOOL
IND

9.1.13.4 Fixity

Binder: ϵ λ
Right Infix 20:
$\Rightarrow$
Right Infix 100:
$\rightarrow$
Right Infix 200:
$=$

9.1.13.5 Terminators

ϵ λ $=$ $\Rightarrow$ $\rightarrow$

9.1.14 The Theory misc

9.1.14.1 Parents

init

9.1.14.2 Children

pair

9.1.14.3 Constants

Arbitrary	$'a$
$\$\exists_1$	$('a \to BOOL) \to BOOL$
Let	$('a \to 'b) \to 'a \to 'b$
Cond	$BOOL \to 'a \to 'a \to 'a$
pp′TS	$BOOL \to BOOL$

9.1.14.4 Aliases

$\Leftrightarrow$	$\$= \; : \; BOOL \to BOOL \to BOOL$

9.1.14.5 Fixity

Binder: $\exists_1$
Right Infix 10:
$\Leftrightarrow$

9.1.14.6 Terminators

$\Leftrightarrow \qquad \exists_1$

9.1.14.7 Definitions

$\exists_1$	
$\exists_1$**_def**	$\vdash \$\exists_1 = (\lambda\ P\bullet\ \exists\ t\bullet\ P\ t \land (\forall\ x\bullet\ P\ x \Rightarrow x = t))$
Let	
let_def	$\vdash Let = (\lambda\ f\ x\bullet\ f\ x)$
Cond	
cond_def	$\vdash Cond$ $= (\lambda\ b\ x1\ x2$ $\bullet\ \epsilon\ x\bullet\ ((b \Leftrightarrow T) \Rightarrow x = x1) \land ((b \Leftrightarrow F) \Rightarrow x = x2))$
pp′TS	
pp′ts_def	$\vdash \forall\ x\bullet\ pp'TS\ x \Leftrightarrow x$

9.1.14.8 Theorems

t_thm	$\vdash T$
∨_thm	$\vdash \forall\ t1\ t2\bullet\ t1 \lor t2 \Leftrightarrow (\forall\ b\bullet\ (t1 \Rightarrow b) \Rightarrow (t2 \Rightarrow b) \Rightarrow b)$
¬_thm	$\vdash \forall\ t\bullet\ \neg\ t \Leftrightarrow t \Rightarrow F$
f_thm	$\vdash \neg\ F$
¬_t_thm	$\vdash \neg\ T \Leftrightarrow F$
∧_thm	$\vdash \forall\ t1\ t2\bullet\ t1 \land t2 \Leftrightarrow (\forall\ b\bullet\ (t1 \Rightarrow t2 \Rightarrow b) \Rightarrow b)$
¬_thm1	$\vdash \forall\ t\bullet\ \neg\ t \Leftrightarrow t \Leftrightarrow F$
∃_intro_thm	$\vdash \forall\ P\ x\bullet\ P\ x \Rightarrow \$\exists\ P$
cond_thm	$\vdash \forall\ a\ t1\ t2$ $\bullet\ (if\ a\ then\ t1\ else\ t2)$ $= (\epsilon\ x\bullet\ ((a \Leftrightarrow T) \Rightarrow x = t1) \land ((a \Leftrightarrow F) \Rightarrow x = t2))$
¬_∀_thm	$\vdash \forall\ p\bullet\ \neg\ \$\forall\ p \Leftrightarrow (\exists\ x\bullet\ \neg\ p\ x)$
¬_∃_thm	$\vdash \forall\ p\bullet\ \neg\ \$\exists\ p \Leftrightarrow (\forall\ x\bullet\ \neg\ p\ x)$
∃₁_thm	$\vdash \forall\ P\bullet\ \$\exists_1\ P \Leftrightarrow (\exists\ t\bullet\ P\ t \land (\forall\ x\bullet\ P\ x \Rightarrow x = t))$
⇔_thm	$\vdash \forall\ a\ b\bullet\ (a \Leftrightarrow b) \Leftrightarrow (a \Rightarrow b) \land (b \Rightarrow a)$
⇒_thm	$\vdash \forall\ a\ b\bullet\ a \Rightarrow b \Leftrightarrow \neg\ a \lor b$
¬_¬_thm	$\vdash \forall\ a\bullet\ \neg\ \neg\ a \Leftrightarrow a$
¬_∨_thm	$\vdash \forall\ a\ b\bullet\ \neg\ (a \lor b) \Leftrightarrow \neg\ a \land \neg\ b$
¬_∧_thm	$\vdash \forall\ a\ b\bullet\ \neg\ (a \land b) \Leftrightarrow \neg\ a \lor \neg\ b$
¬_⇒_thm	$\vdash \forall\ a\ b\bullet\ \neg\ (a \Rightarrow b) \Leftrightarrow a \land \neg\ b$
¬_⇔_thm	$\vdash \forall\ a\ b\bullet\ \neg\ (a \Leftrightarrow b) \Leftrightarrow a \land \neg\ b \lor b \land \neg\ a$
¬_f_thm	$\vdash \neg\ F \Leftrightarrow T$
¬_if_thm	$\vdash \forall\ a\ b\ c$ $\bullet\ \neg\ (if\ a\ then\ b\ else\ c) \Leftrightarrow (if\ a\ then\ \neg\ b\ else\ \neg\ c)$
if_thm	$\vdash \forall\ a\ b\ c\bullet\ (if\ a\ then\ b\ else\ c) \Leftrightarrow a \land b \lor \neg\ a \land c$
eq_rewrite_thm	$\vdash \forall\ x\bullet\ x = x \Leftrightarrow T$
⇔_rewrite_thm	$\vdash \forall\ t$ $\bullet\ ((T \Leftrightarrow t) \Leftrightarrow t)$ $\land\ ((t \Leftrightarrow T) \Leftrightarrow t)$ $\land\ ((F \Leftrightarrow t) \Leftrightarrow \neg\ t)$ $\land\ ((t \Leftrightarrow F) \Leftrightarrow \neg\ t)$
¬_rewrite_thm	$\vdash \forall\ t\bullet\ (\neg\ \neg\ t \Leftrightarrow t) \land (\neg\ T \Leftrightarrow F) \land (\neg\ F \Leftrightarrow T)$
∧_rewrite_thm	$\vdash \forall\ t$ $\bullet\ (T \land t \Leftrightarrow t)$ $\land\ (t \land T \Leftrightarrow t)$ $\land\ \neg\ (F \land t)$ $\land\ \neg\ (t \land F)$ $\land\ (t \land t \Leftrightarrow t)$
∨_rewrite_thm	$\vdash \forall\ t$ $\bullet\ (T \lor t)$ $\land\ (t \lor T)$ $\land\ (F \lor t \Leftrightarrow t)$ $\land\ (t \lor F \Leftrightarrow t)$ $\land\ (t \lor t \Leftrightarrow t)$
⇒_rewrite_thm	

$\vdash \forall\ t$
$\quad \bullet\ (T \Rightarrow t \Leftrightarrow t)$
$\qquad \wedge\ (F \Rightarrow t \Leftrightarrow T)$
$\qquad \wedge\ (t \Rightarrow T \Leftrightarrow T)$
$\qquad \wedge\ (t \Rightarrow t \Leftrightarrow T)$
$\qquad \wedge\ (t \Rightarrow F \Leftrightarrow \neg\ t)$

if_rewrite_thm
$\vdash \forall\ t1\ t2$
$\quad \bullet\ (if\ T\ then\ t1\ else\ t2) = t1$
$\qquad \wedge\ (if\ F\ then\ t1\ else\ t2) = t2$

∀_rewrite_thm
$\vdash \forall\ t\bullet\ (\forall\ x\bullet\ t) \Leftrightarrow t$

∃_rewrite_thm
$\vdash \forall\ t\bullet\ (\exists\ x\bullet\ t) \Leftrightarrow t$

β_rewrite_thm
$\vdash \forall\ t1\ t2\bullet\ (\lambda\ x\bullet\ t1)\ t2 = t1$

one_one_thm $\vdash \forall\ f\bullet\ OneOne\ f \Leftrightarrow (\forall\ x\ y\bullet\ f\ x = f\ y \Leftrightarrow x = y)$

ext_thm $\vdash \forall\ f\ g\bullet\ f = g \Leftrightarrow (\forall\ x\bullet\ f\ x = g\ x)$

type_lemmas_thm
$\vdash \forall\ pred$
$\quad \bullet\ (\exists\ f\bullet\ TypeDefn\ pred\ f)$
$\qquad \Rightarrow (\exists\ abs\ rep$
$\qquad \bullet\ (\forall\ a\bullet\ abs\ (rep\ a) = a)$
$\qquad\quad \wedge\ (\forall\ r\bullet\ pred\ r \Leftrightarrow rep\ (abs\ r) = r))$

fun_rel_thm $\vdash \forall\ r$
$\quad \bullet\ (\exists\ f\bullet\ \forall\ x\ y\bullet\ f\ x = y \Leftrightarrow r\ x\ y)$
$\qquad \Leftrightarrow (\forall\ x\bullet\ \exists\ y\bullet\ r\ x\ y \wedge (\forall\ z\bullet\ r\ x\ z \Rightarrow z = y))$

9.1.15 The Theory one

9.1.15.1 Parents

basic_hol

9.1.15.2 Children

hol

9.1.15.3 Constants

IsOneRep	$BOOL \rightarrow BOOL$
One	ONE

9.1.15.4 Types

ONE

9.1.15.5 Definitions

IsOneRep
is_one_rep_def
$\vdash \exists\ one \bullet\ \forall\ x \bullet\ IsOneRep\ x \Leftrightarrow x \Leftrightarrow one$
ONE
one_def1 $\vdash \exists\ f \bullet\ TypeDefn\ IsOneRep\ f$
One
one_def $\vdash \forall\ x \bullet\ x = One$

9.1.15.6 Theorems

one_fns_thm $\vdash \forall\ f \bullet\ f = (\lambda\ x \bullet\ One)$

9.1.16 The Theory orders

9.1.16.1 Parents

sets

9.1.16.2 Children

$\mathbb{R}$

9.1.16.3 Constants

Irrefl	$'a\ SET \times ('a \to 'a \to BOOL) \to BOOL$
Antisym	$'a\ SET \times ('a \to 'a \to BOOL) \to BOOL$
Trans	$'a\ SET \times ('a \to 'a \to BOOL) \to BOOL$
StrictPartialOrder	$'a\ SET \times ('a \to 'a \to BOOL) \to BOOL$
Trich	$'a\ SET \times ('a \to 'a \to BOOL) \to BOOL$
StrictLinearOrder	$'a\ SET \times ('a \to 'a \to BOOL) \to BOOL$
$DenseIn	$'a\ SET \to 'a\ SET \times ('a \to 'a \to BOOL) \to BOOL$
Dense	$'a\ SET \times ('a \to 'a \to BOOL) \to BOOL$
UpperBound	$'a\ SET \times ('a \to 'a \to BOOL) \times 'a \to BOOL$
$HasSupremum	$'a\ SET \to 'a \times 'a\ SET \times ('a \to 'a \to BOOL) \to BOOL$
UnboundedAbove	$'a\ SET \times ('a \to 'a \to BOOL) \to BOOL$
UnboundedBelow	$'a\ SET \times ('a \to 'a \to BOOL) \to BOOL$
Complete	$'a\ SET \times ('a \to 'a \to BOOL) \to BOOL$
Cuts	$'a\ SET \times ('a \to 'a \to BOOL) \to 'a\ SET\ SET$
DownSet	$'a\ SET \times ('a \to 'a \to BOOL) \times 'a \to 'a\ SET$
DownSets	$'a\ SET \times ('a \to 'a \to BOOL) \times 'a\ SET \to 'a\ SET\ SET$

9.1.16.4 Fixity

Right Infix 210:

DenseIn **HasSupremum** << <<<

9.1.16.5 Definitions

Irrefl $\vdash \forall\ X\ \$<<\bullet\ Irrefl\ (X, \$<<) \Leftrightarrow (\forall\ x\bullet\ x \in X \Rightarrow \neg\ x << x)$

Antisym
$$\begin{array}{l} \vdash \forall\ X\ \$<< \\ \quad \bullet\ Antisym\ (X, \$<<) \\ \qquad \Leftrightarrow (\forall\ x\ y \\ \qquad \bullet\ x \in X \wedge y \in X \wedge \neg\ x = y \Rightarrow \neg\ (x << y \wedge y << x)) \end{array}$$

Trans
$$\begin{array}{l} \vdash \forall\ X\ \$<< \\ \quad \bullet\ Trans\ (X, \$<<) \\ \qquad \Leftrightarrow (\forall\ x\ y\ z \\ \qquad \bullet\ x \in X \wedge y \in X \wedge z \in X \wedge x << y \wedge y << z \\ \qquad\quad \Rightarrow x << z) \end{array}$$

StrictPartialOrder

$$\begin{array}{l}
\vdash \forall\ X\ \$<< \\
\quad \bullet\ \mathit{StrictPartialOrder}\ (X,\ \$<<) \\
\qquad \Leftrightarrow \mathit{Irrefl}\ (X,\ \$<<) \\
\qquad\quad \wedge\ \mathit{Antisym}\ (X,\ \$<<) \\
\qquad\quad \wedge\ \mathit{Trans}\ (X,\ \$<<)
\end{array}$$

Trich

$$\begin{array}{l}
\vdash \forall\ X\ \$<< \\
\quad \bullet\ \mathit{Trich}\ (X,\ \$<<) \\
\qquad \Leftrightarrow (\forall\ x\ y \\
\qquad \bullet\ x \in X \wedge y \in X \wedge \neg\ x = y \Rightarrow x << y \vee y << x)
\end{array}$$

StrictLinearOrder

$$\begin{array}{l}
\vdash \forall\ X\ \$<< \\
\quad \bullet\ \mathit{StrictLinearOrder}\ (X,\ \$<<) \\
\qquad \Leftrightarrow \mathit{StrictPartialOrder}\ (X,\ \$<<) \wedge \mathit{Trich}\ (X,\ \$<<)
\end{array}$$

DenseIn

$$\begin{array}{l}
\vdash \forall\ A\ X\ \$<< \\
\quad \bullet\ A\ \mathit{DenseIn}\ (X,\ \$<<) \\
\qquad \Leftrightarrow (\forall\ x\ y \\
\qquad \bullet\ x \in X \wedge y \in X \wedge x << y \\
\qquad\qquad \Rightarrow (\exists\ a \bullet\ a \in A \wedge x << a \wedge a << y))
\end{array}$$

Dense

$$\vdash \forall\ X\ \$<< \bullet\ \mathit{Dense}\ (X,\ \$<<) \Leftrightarrow X\ \mathit{DenseIn}\ (X,\ \$<<)$$

UpperBound

$$\begin{array}{l}
\vdash \forall\ A\ \$<<\ x \\
\quad \bullet\ \mathit{UpperBound}\ (A,\ \$<<,\ x) \Leftrightarrow (\forall\ a \bullet\ a \in A \Rightarrow a << x)
\end{array}$$

HasSupremum

$$\begin{array}{l}
\vdash \forall\ A\ \$<<\ x\ X \\
\quad \bullet\ A\ \mathit{HasSupremum}\ (x,\ X,\ \$<<) \\
\qquad \Leftrightarrow \mathit{UpperBound}\ (A,\ \$<<,\ x) \\
\qquad\quad \wedge\ (\forall\ y \\
\qquad\quad \bullet\ y \in X \wedge \mathit{UpperBound}\ (A,\ \$<<,\ y) \Rightarrow \neg\ y << x)
\end{array}$$

UnboundedAbove

$$\begin{array}{l}
\vdash \forall\ A\ \$<< \\
\quad \bullet\ \mathit{UnboundedAbove}\ (A,\ \$<<) \\
\qquad \Leftrightarrow (\forall\ b \bullet\ b \in A \Rightarrow (\exists\ c \bullet\ c \in A \wedge b << c))
\end{array}$$

UnboundedBelow

$$\begin{array}{l}
\vdash \forall\ A\ \$<< \\
\quad \bullet\ \mathit{UnboundedBelow}\ (A,\ \$<<) \\
\qquad \Leftrightarrow (\forall\ b \bullet\ b \in A \Rightarrow (\exists\ c \bullet\ c \in A \wedge c << b))
\end{array}$$

Complete

$$\begin{array}{l}
\vdash \forall\ X\ \$<< \\
\quad \bullet\ \mathit{Complete}\ (X,\ \$<<) \\
\qquad \Leftrightarrow (\forall\ A\ x \\
\qquad \bullet\ \neg\ A = \{\} \\
\qquad\qquad\quad \wedge\ A \subseteq X \\
\qquad\qquad\quad \wedge\ \mathit{UnboundedAbove}\ (A,\ \$<<) \\
\qquad\qquad\quad \wedge\ x \in X \\
\qquad\qquad\quad \wedge\ \mathit{UpperBound}\ (A,\ \$<<,\ x) \\
\qquad\qquad \Rightarrow (\exists\ y \bullet\ y \in X \wedge A\ \mathit{HasSupremum}\ (y,\ X,\ \$<<)))
\end{array}$$

Cuts

$$\begin{array}{l}
\vdash \forall\ X\ \$<<\ A \\
\quad \bullet\ A \in \mathit{Cuts}\ (X,\ \$<<) \\
\qquad \Leftrightarrow A \subseteq X \\
\qquad\quad \wedge\ \neg\ A = \{\} \\
\qquad\quad \wedge\ \mathit{UnboundedAbove}\ (A,\ \$<<) \\
\qquad\quad \wedge\ (\exists\ x \bullet\ x \in X \wedge \mathit{UpperBound}\ (A,\ \$<<,\ x)) \\
\qquad\quad \wedge\ (\forall\ a\ b \bullet\ a \in A \wedge b \in X \wedge b << a \Rightarrow b \in A)
\end{array}$$

DownSet $\vdash \forall\ X\ \$<<\ x \bullet\ DownSet\ (X,\ \$<<,\ x) = \{y | y \in X \land y << x\}$

DownSets $\vdash \forall\ X\ \$<<\ A\ s$
$\bullet\ s \in DownSets\ (X,\ \$<<,\ A)$
$\Leftrightarrow (\exists\ x \bullet\ x \in A \land s = DownSet\ (X,\ \$<<,\ x))$

9.1.16.6 Theorems

⊂_irrefl_thm $\vdash \forall\ V \bullet\ Irrefl\ (V,\ \$\subset)$

⊂_antisym_thm
$\vdash \forall\ V \bullet\ Antisym\ (V,\ \$\subset)$

⊂_trans_thm $\vdash \forall\ V \bullet\ Trans\ (V,\ \$\subset)$

cuts_trich_thm
$\vdash \forall\ X\ \$<<$
$\bullet\ Trans\ (X,\ \$<<) \land Trich\ (X,\ \$<<)$
$\Rightarrow Trich\ (Cuts\ (X,\ \$<<),\ \$\subset)$

cuts_strict_partial_order_thm
$\vdash \forall\ X\ \$<< \bullet\ StrictPartialOrder\ (Cuts\ (X,\ \$<<),\ \$\subset)$

cuts_strict_linear_order_thm
$\vdash \forall\ X\ \$<<$
$\bullet\ StrictLinearOrder\ (X,\ \$<<)$
$\Rightarrow StrictLinearOrder\ (Cuts\ (X,\ \$<<),\ \$\subset)$

cuts_complete_thm
$\vdash \forall\ X\ \$<< \bullet\ Complete\ (Cuts\ (X,\ \$<<),\ \$\subset)$

down_sets_cuts_thm
$\vdash \forall\ X\ \$<<\ A$
$\bullet\ A \subseteq X$
$\land\ Irrefl\ (X,\ \$<<)$
$\land\ Trans\ (X,\ \$<<)$
$\land\ UnboundedBelow\ (A,\ \$<<)$
$\land\ A\ DenseIn\ (X,\ \$<<)$
$\Rightarrow DownSets\ (X,\ \$<<,\ A) \subseteq Cuts\ (X,\ \$<<)$

down_sets_dense_thm
$\vdash \forall\ X\ \$<<\ A$
$\bullet\ A \subseteq X$
$\land\ StrictLinearOrder\ (X,\ \$<<)$
$\land\ UnboundedBelow\ (A,\ \$<<)$
$\land\ A\ DenseIn\ (X,\ \$<<)$
$\Rightarrow DownSets\ (X,\ \$<<,\ A)$
$DenseIn\ (Cuts\ (X,\ \$<<),\ \$\subset)$

dense_superset_thm
$\vdash \forall\ X\ \$<<\ A\ B$
$\bullet\ A \subseteq B \land B \subseteq X \land A\ DenseIn\ (X,\ \$<<)$
$\Rightarrow B\ DenseIn\ (X,\ \$<<)$

dense_universe_thm
$\vdash \forall\ X\ \$<<\ A$
$\bullet\ A \subseteq X \land A\ DenseIn\ (X,\ \$<<) \Rightarrow Dense\ (X,\ \$<<)$

downset_cut_thm
$\vdash \forall\ X\ \$<<\ a$
$\bullet\ a \in X$
$\land\ Trans\ (X,\ \$<<)$
$\land\ UnboundedBelow\ (X,\ \$<<)$

$$\begin{array}{l}
\quad\quad \land\ X\ \mathit{DenseIn}\ (X,\ \$<<) \\
\quad \Rightarrow \mathit{DownSet}\ (X,\ \$<<,\ a) \in \mathit{Cuts}\ (X,\ \$<<)
\end{array}$$

down_sets_less_thm

$$\begin{array}{l}
\vdash \forall\ X\ \$<<\ a\ b \\
\quad \bullet\ a \in X \land b \in X \land \mathit{StrictLinearOrder}\ (X,\ \$<<) \\
\quad\quad \Rightarrow (\mathit{DownSet}\ (X,\ \$<<,\ a) \subset \mathit{DownSet}\ (X,\ \$<<,\ b) \\
\quad\quad\quad \Leftrightarrow a << b)
\end{array}$$

cuts_unbounded_above_thm

$$\begin{array}{l}
\vdash \forall\ X\ \$<< \\
\quad \bullet\ \mathit{Irrefl}\ (X,\ \$<<) \\
\quad\quad \land\ \mathit{Trans}\ (X,\ \$<<) \\
\quad\quad \land\ \mathit{UnboundedAbove}\ (X,\ \$<<) \\
\quad\quad \land\ X\ \mathit{DenseIn}\ (X,\ \$<<) \\
\quad \Rightarrow \mathit{UnboundedAbove}\ (\mathit{Cuts}\ (X,\ \$<<),\ \$\subset)
\end{array}$$

cuts_unbounded_below_thm

$$\begin{array}{l}
\vdash \forall\ X\ \$<< \\
\quad \bullet\ \mathit{Irrefl}\ (X,\ \$<<) \\
\quad\quad \land\ \mathit{Trans}\ (X,\ \$<<) \\
\quad\quad \land\ \mathit{UnboundedBelow}\ (X,\ \$<<) \\
\quad\quad \land\ X\ \mathit{DenseIn}\ (X,\ \$<<) \\
\quad \Rightarrow \mathit{UnboundedBelow}\ (\mathit{Cuts}\ (X,\ \$<<),\ \$\subset)
\end{array}$$

dense_complete_subset_thm

$$\begin{array}{l}
\vdash \forall\ X\ \$<<\ A \\
\quad \bullet\ \mathit{StrictLinearOrder}\ (X,\ \$<<) \\
\quad\quad \land\ A \subseteq X \\
\quad\quad \land\ \mathit{UnboundedAbove}\ (X,\ \$<<) \\
\quad\quad \land\ \mathit{UnboundedBelow}\ (X,\ \$<<) \\
\quad\quad \land\ A\ \mathit{DenseIn}\ (X,\ \$<<) \\
\quad\quad \land\ \mathit{Complete}\ (A,\ \$<<) \\
\quad \Rightarrow A = X
\end{array}$$

induced_order_irrefl_thm

$$\begin{array}{l}
\vdash \forall\ X\ \$<<\ f \\
\quad \bullet\ (\forall\ a \bullet\ f\ a \in X) \land \mathit{Irrefl}\ (X,\ \$<<) \\
\quad\quad \Rightarrow \mathit{Irrefl}\ (\mathit{Universe},\ (\lambda\ a\ b \bullet\ f\ a << f\ b))
\end{array}$$

induced_order_antisym_thm

$$\begin{array}{l}
\vdash \forall\ X\ \$<<\ f \\
\quad \bullet\ (\forall\ a \bullet\ f\ a \in X) \land \mathit{Irrefl}\ (X,\ \$<<) \land \mathit{Antisym}\ (X,\ \$<<) \\
\quad\quad \Rightarrow \mathit{Antisym}\ (\mathit{Universe},\ (\lambda\ a\ b \bullet\ f\ a << f\ b))
\end{array}$$

induced_order_trans_thm

$$\begin{array}{l}
\vdash \forall\ X\ \$<<\ f \\
\quad \bullet\ (\forall\ a \bullet\ f\ a \in X) \land \mathit{Trans}\ (X,\ \$<<) \\
\quad\quad \Rightarrow \mathit{Trans}\ (\mathit{Universe},\ (\lambda\ a\ b \bullet\ f\ a << f\ b))
\end{array}$$

induced_order_trich_thm

$$\begin{array}{l}
\vdash \forall\ X\ \$<<\ f \\
\quad \bullet\ (\forall\ a \bullet\ f\ a \in X) \land \mathit{OneOne}\ f \land \mathit{Trich}\ (X,\ \$<<) \\
\quad\quad \Rightarrow \mathit{Trich}\ (\mathit{Universe},\ (\lambda\ a\ b \bullet\ f\ a << f\ b))
\end{array}$$

induced_strict_partial_order_thm

$$\begin{array}{l}
\vdash \forall\ X\ \$<<\ f \\
\quad \bullet\ (\forall\ a \bullet\ f\ a \in X) \land \mathit{StrictPartialOrder}\ (X,\ \$<<) \\
\quad\quad \Rightarrow \mathit{StrictPartialOrder} \\
\quad\quad\quad (\mathit{Universe},\ (\lambda\ a\ b \bullet\ f\ a << f\ b))
\end{array}$$

induced_strict_linear_order_thm

$\vdash \forall\ X\ \${<}{<}\ f$
$\quad \bullet\ (\forall\ a \bullet\ f\ a \in X)$
$\qquad \wedge\ \mathit{OneOne}\ f$
$\qquad \wedge\ \mathit{StrictLinearOrder}\ (X,\ \${<}{<})$
$\qquad \Rightarrow\ \mathit{StrictLinearOrder}$
$\qquad (\mathit{Universe},\ (\lambda\ a\ b \bullet\ f\ a << f\ b))$

induced_order_complete_thm

$\vdash \forall\ X\ \${<}{<}\ f$
$\quad \bullet\ (\forall\ a \bullet\ f\ a \in X)$
$\qquad \wedge\ \mathit{OneOne}\ f$
$\qquad \wedge\ (\forall\ x \bullet\ x \in X \Rightarrow (\exists\ a \bullet\ x = f\ a))$
$\qquad \wedge\ \mathit{Complete}\ (X,\ \${<}{<})$
$\qquad \Rightarrow\ \mathit{Complete}\ (\mathit{Universe},\ (\lambda\ a\ b \bullet\ f\ a << f\ b))$

induced_order_dense_thm

$\vdash \forall\ X\ \${<}{<}\ f\ A$
$\quad \bullet\ (\forall\ a \bullet\ f\ a \in X)$
$\qquad \wedge\ \mathit{OneOne}\ f$
$\qquad \wedge\ (\forall\ x \bullet\ x \in X \Rightarrow (\exists\ a \bullet\ x = f\ a))$
$\qquad \wedge\ \{x | \exists\ a \bullet\ a \in A \wedge x = f\ a\}\ \mathit{DenseIn}\ (X,\ \${<}{<})$
$\qquad \Rightarrow\ A\ \mathit{DenseIn}\ (\mathit{Universe},\ (\lambda\ a\ b \bullet\ f\ a << f\ b))$

induced_order_unbounded_above_thm

$\vdash \forall\ X\ \${<}{<}\ f$
$\quad \bullet\ (\forall\ a \bullet\ f\ a \in X)$
$\qquad \wedge\ \mathit{OneOne}\ f$
$\qquad \wedge\ (\forall\ x \bullet\ x \in X \Rightarrow (\exists\ a \bullet\ x = f\ a))$
$\qquad \wedge\ \mathit{UnboundedAbove}\ (X,\ \${<}{<})$
$\qquad \Rightarrow\ \mathit{UnboundedAbove}\ (\mathit{Universe},\ (\lambda\ a\ b \bullet\ f\ a << f\ b))$

induced_order_unbounded_below_thm

$\vdash \forall\ X\ \${<}{<}\ f$
$\quad \bullet\ (\forall\ a \bullet\ f\ a \in X)$
$\qquad \wedge\ \mathit{OneOne}\ f$
$\qquad \wedge\ (\forall\ x \bullet\ x \in X \Rightarrow (\exists\ a \bullet\ x = f\ a))$
$\qquad \wedge\ \mathit{UnboundedBelow}\ (X,\ \${<}{<})$
$\qquad \Rightarrow\ \mathit{UnboundedBelow}\ (\mathit{Universe},\ (\lambda\ a\ b \bullet\ f\ a << f\ b))$

9.1.17 The Theory pair

9.1.17.1 Parents

misc

9.1.17.2 Children

$\mathbb{N}$

9.1.17.3 Constants

IsPairRep	$('a \to 'b \to BOOL) \to BOOL$
Snd	$'a \times 'b \to 'b$
Fst	$'a \times 'b \to 'a$
$,	$'a \to 'b \to 'a \times 'b$
Uncurry	$('a \to 'b \to 'c) \to 'a \times 'b \to 'c$
Curry	$('a \times 'b \to 'c) \to 'a \to 'b \to 'c$

9.1.17.4 Types

$'1 \times '2$

9.1.17.5 Fixity

Right Infix 150:

, $\times$

9.1.17.6 Terminators

$\times$

9.1.17.7 Definitions

IsPairRep
is_pair_rep_def

$$\begin{array}{l} \vdash \exists\ comma\ fst\ snd \\ \quad \bullet\ (\forall\ x\ y \\ \qquad \bullet\ IsPairRep\ (comma\ x\ y) \\ \qquad\quad \land\ fst\ (comma\ x\ y) = x \\ \qquad\quad \land\ snd\ (comma\ x\ y) = y) \\ \qquad \land\ (\forall\ x\ y\ a\ b \\ \qquad \bullet\ comma\ a\ b = comma\ x\ y \Leftrightarrow a = x \land b = y) \\ \qquad \land\ (\forall\ p\bullet\ IsPairRep\ p \Rightarrow comma\ (fst\ p)\ (snd\ p) = p) \end{array}$$

$\times$
$\times$_def $\vdash \exists\ f\bullet\ TypeDefn\ IsPairRep\ f$

,
Fst
Snd
pair_ops_def $\vdash (\forall\ x\ y\bullet\ Fst\ (x,\ y) = x \land Snd\ (x,\ y) = y)$

$\land$ ($\forall$ x y a $b\bullet$ $(a,\ b) = (x,\ y) \Leftrightarrow a = x \land b = y$)
$\land$ ($\forall$ $p\bullet$ $(Fst\ p,\ Snd\ p) = p$)

Uncurry

uncurry_def $\vdash \forall\ f\ x\ y\bullet\ Uncurry\ f\ (x,\ y) = f\ x\ y$

Curry

curry_def $\vdash \forall\ f\ x\ y\bullet\ Curry\ f\ x\ y = f\ (x,\ y)$

9.1.17.8 Theorems

pair_clauses $\vdash \forall\ x\ y\ a\ b\ p\ fu\ fc$
$\bullet\ Fst\ (x,\ y) = x$
$\land\ Snd\ (x,\ y) = y$
$\land\ ((a,\ b) = (x,\ y) \Leftrightarrow a = x \land b = y)$
$\land\ (Fst\ p,\ Snd\ p) = p$
$\land\ Curry\ fc\ x\ y = fc\ (x,\ y)$
$\land\ Uncurry\ fu\ (x,\ y) = fu\ x\ y$
$\land\ Uncurry\ fu\ p = fu\ (Fst\ p)\ (Snd\ p)$
$\land\ ((a,\ b) = p \Leftrightarrow a = Fst\ p \land b = Snd\ p)$
$\land\ (p = (a,\ b) \Leftrightarrow Fst\ p = a \land Snd\ p = b)$

9.1.18 The Theory seq

9.1.18.1 Parents

fun_rel

9.1.18.2 Children

fin_set

9.1.18.3 Constants

Elems	$'a\ LIST \to 'a\ SET$
Distinct	$'a\ LIST\ SET$
Lists	$'a\ SET \to 'a\ LIST\ SET$
Lists$_1$	$'a\ SET \to 'a\ LIST\ SET$
InjLists	$'a\ SET \to 'a\ LIST\ SET$
Nth	$'a\ LIST \to \mathbb{N} \to 'a$
$..	$\mathbb{N} \to \mathbb{N} \to \mathbb{N}\ SET$
ListRel	$'a\ LIST \to (\mathbb{N} \times 'a)\ SET$
RelList	$(\mathbb{N} \times 'a)\ SET \to 'a\ LIST$
Last	$'a\ LIST \to 'a$
Front	$'a\ LIST \to 'a\ LIST$
$$\upharpoonright$	$'a\ LIST \to 'a\ SET \to 'a\ LIST$
Enumerate	$\mathbb{N}\ SET \to (\mathbb{N} \times \mathbb{N})\ SET$
Squash	$(\mathbb{N} \times 'a)\ SET \to (\mathbb{N} \times 'a)\ SET$
Extract	$\mathbb{N}\ SET \to 'a\ LIST \to 'a\ LIST$
$**Prefix**	$'a\ LIST \to 'a\ LIST \to BOOL$
$**Suffix**	$'a\ LIST \to 'a\ LIST \to BOOL$
$**In**	$'a\ LIST \to 'a\ LIST \to BOOL$
Flat	$'a\ LIST\ LIST \to 'a\ LIST$

9.1.18.4 Aliases

#	$Length : 'a\ LIST \to \mathbb{N}$
$\frown$	$Append : 'a\ LIST \to 'a\ LIST \to 'a\ LIST$
Head	$Hd : 'a\ LIST \to 'a$
Tail	$Tl : 'a\ LIST \to 'a\ LIST$

9.1.18.5 Fixity

Left Infix 300:

$\upharpoonright$

Right Infix 290:

..

Right Infix 300:

In **Prefix** **Suffix** $\frown$

9.1.18.6 Definitions

Elems $\vdash$ *Elems* [] = {}
$\wedge (\forall\ x\ l\bullet\ Elems\ (Cons\ x\ l) = \{x\} \cup Elems\ l)$

Distinct $\vdash [] \in Distinct$
$\wedge (\forall\ x\ l$
$\bullet\ Cons\ x\ l \in Distinct$
$\Leftrightarrow \neg\ x \in Elems\ l \wedge l \in Distinct)$

Lists $\vdash \forall\ a\bullet\ Lists\ a = \{l | Elems\ l \subseteq a\}$

Lists₁ $\vdash \forall\ a\bullet\ Lists_1\ a = Lists\ a \setminus \{[]\}$

InjLists $\vdash \forall\ a\bullet\ InjLists\ a = Lists\ a \cap Distinct$

Nth $\vdash \forall\ l\ x\ n$
$\bullet\ Nth\ (Cons\ x\ l)\ n$
$= (if\ n = 1\ then\ x\ else\ Nth\ l\ (n - 1))$

.. $\vdash \forall\ m\ n\bullet\ m\ ..\ n = \{i | m \le i \wedge i \le n\}$

ListRel $\vdash \forall\ l\bullet\ ListRel\ l = 1\ ..\ \#\ l \lhd Graph\ (Nth\ l)$

RelList $\vdash ConstSpec$
$(\lambda\ RelList'\bullet\ \forall\ l\bullet\ RelList'\ (ListRel\ l) = l)$
$RelList$

Last $\vdash \forall\ x\ l$
$\bullet\ Last\ (Cons\ x\ l) = (if\ l = []\ then\ x\ else\ Last\ l)$

Front $\vdash \forall\ x\ l$
$\bullet\ Front\ (Cons\ x\ l)$
$= (if\ l = []\ then\ []\ else\ Cons\ x\ (Front\ l))$

↾ $\vdash \forall\ a\ x\ l$
$\bullet\ [] \upharpoonright a = []$
$\wedge\ Cons\ x\ l \upharpoonright a$
$= (if\ x \in a\ then\ Cons\ x\ (l \upharpoonright a)\ else\ l \upharpoonright a)$

Enumerate $\vdash \forall\ a$
$\bullet\ Enumerate\ a$
$= \{(i, j)$
$| \exists\ l$
$\bullet\ j \in a$
$\wedge\ l \in Distinct$
$\wedge\ \#\ l = i$
$\wedge\ Elems\ l = a \cap 0\ ..\ j\}$

Squash $\vdash \forall\ r\bullet\ Squash\ r = Enumerate\ (Dom\ r)\ \fatsemi\ r$

Extract $\vdash \forall\ a\ l\bullet\ Extract\ a\ l = RelList\ (Squash\ (a \lhd ListRel\ l))$

Prefix $\vdash \forall\ s\ t\bullet\ s\ Prefix\ t \Leftrightarrow (\exists\ v\bullet\ s \frown v = t)$

Suffix $\vdash \forall\ s\ t\bullet\ s\ Suffix\ t \Leftrightarrow (\exists\ u\bullet\ u \frown s = t)$

In $\vdash \forall\ s\ t\bullet\ s\ In\ t \Leftrightarrow (\exists\ u\ v\bullet\ u \frown s \frown v = t)$

Flat $\vdash \forall\ e\ l\bullet\ Flat\ [] = [] \wedge Flat\ (Cons\ e\ l) = e \frown Flat\ l$

9.1.19 The Theory sets

9.1.19.1 Parents

basic_hol

9.1.19.2 Children

orders *set_thms* $\mathbb{Z}$ *hol*

9.1.19.3 Constants

IsSetRep	$('a \to BOOL) \to BOOL$
$\in	$'a \to 'a\ SET \to BOOL$
$**SetComp**	$('a \to BOOL) \to 'a\ SET$
Insert	$'a \to 'a\ SET \to 'a\ SET$
Universe	$'a\ SET$
{}	$'a\ SET$
∼	$'a\ SET \to 'a\ SET$
$∪	$'a\ SET \to 'a\ SET \to 'a\ SET$
$∩	$'a\ SET \to 'a\ SET \to 'a\ SET$
$\	$'a\ SET \to 'a\ SET \to 'a\ SET$
$⊖	$'a\ SET \to 'a\ SET \to 'a\ SET$
$⊆	$'a\ SET \to 'a\ SET \to BOOL$
$⊂	$'a\ SET \to 'a\ SET \to BOOL$
⋃	$'a\ SET\ SET \to 'a\ SET$
⋂	$'a\ SET\ SET \to 'a\ SET$
ℙ	$'a\ SET \to 'a\ SET\ SET$

9.1.19.4 Types

'1 **SET**

9.1.19.5 Fixity

Binder: **SetComp**

Left Infix 265:

\

Right Infix 230:

⊆ ∈ ⊂

Right Infix 250:

⊖

Right Infix 260:

∪

Right Infix 270:

∩

9.1.19.6 Definitions

IsSetRep
is_set_rep_def
$\vdash IsSetRep = (\lambda\ x\bullet\ T)$
SET
set_def $\vdash \exists\ f\bullet\ TypeDefn\ IsSetRep\ f$
SetComp
$\in$
set_comp_def $\vdash \forall\ x\ p\ a\ b$
$\bullet\ (x \in \{v|p\ v\} \Leftrightarrow p\ x)$
$\wedge\ (a = b \Leftrightarrow (\forall\ x\bullet\ x \in a \Leftrightarrow x \in b))$
Empty
Universe
Insert
insert_def $\vdash \forall\ x\ y\ a$
$\bullet\ \neg\ x \in \{\}$
$\wedge\ x \in Universe$
$\wedge\ (x \in Insert\ y\ a \Leftrightarrow x = y \vee x \in a)$
$\sim$
complement_def
$\vdash \forall\ x\ a\bullet\ x \in \sim a \Leftrightarrow \neg\ x \in a$
$\cup$
$\cup$_def $\vdash \forall\ x\ a\ b\bullet\ x \in a \cup b \Leftrightarrow x \in a \vee x \in b$
$\cap$
$\cap$_def $\vdash \forall\ x\ a\ b\bullet\ x \in a \cap b \Leftrightarrow x \in a \wedge x \in b$
$\setminus$
set_dif_def $\vdash \forall\ x\ a\ b\bullet\ x \in a \setminus b \Leftrightarrow x \in a \wedge \neg\ x \in b$
$\ominus$
$\ominus$_def $\vdash \forall\ x\ a\ b\bullet\ x \in a \ominus b \Leftrightarrow \neg\ (x \in a \Leftrightarrow x \in b)$
$\subseteq$
$\subseteq$_def $\vdash \forall\ a\ b\bullet\ a \subseteq b \Leftrightarrow (\forall\ x\bullet\ x \in a \Rightarrow x \in b)$
$\subset$
$\subset$_def $\vdash \forall\ a\ b\bullet\ a \subset b \Leftrightarrow a \subseteq b \wedge (\exists\ x\bullet\ \neg\ x \in a \wedge x \in b)$
$\bigcup$
$\bigcup$_def $\vdash \forall\ x\ a\bullet\ x \in \bigcup a \Leftrightarrow (\exists\ s\bullet\ x \in s \wedge s \in a)$
$\bigcap$
$\bigcap$_def $\vdash \forall\ x\ a\bullet\ x \in \bigcap a \Leftrightarrow (\forall\ s\bullet\ s \in a \Rightarrow x \in s)$
$\mathbb{P}$
$\mathbb{P}$_def $\vdash \forall\ x\ a\bullet\ x \in \mathbb{P}\ a \Leftrightarrow x \subseteq a$

9.1.19.7 Theorems

sets_clauses $\vdash \forall\ x\ y\ p\ q$
$\bullet\ (x \in \{\} \Leftrightarrow F)$
$\wedge\ (x \in Universe \Leftrightarrow T)$
$\wedge\ (x \in \{v|q\} \Leftrightarrow q)$
$\wedge\ (x \in \{v|p\ v\} \Leftrightarrow p\ x)$
$\wedge\ (x \in \{v|v = y\} \Leftrightarrow x = y)$
$\wedge\ (x \in \{y\} \Leftrightarrow x = y)$
complement_clauses

$\vdash (\forall\ x\ a \bullet\ x \in \sim a \Leftrightarrow \neg\ x \in a)$
$\quad \wedge \sim \mathit{Universe} = \{\}$
$\quad \wedge \sim \{\} = \mathit{Universe}$

∪_clauses $\vdash \forall\ a$
$\bullet\ a \cup \{\} = a$
$\quad \wedge \{\} \cup a = a$
$\quad \wedge a \cup \mathit{Universe} = \mathit{Universe}$
$\quad \wedge \mathit{Universe} \cup a = \mathit{Universe}$
$\quad \wedge a \cup a = a$

∩_clauses $\vdash \forall\ a$
$\bullet\ a \cap \{\} = \{\}$
$\quad \wedge \{\} \cap a = \{\}$
$\quad \wedge a \cap \mathit{Universe} = a$
$\quad \wedge \mathit{Universe} \cap a = a$
$\quad \wedge a \cap a = a$

set_dif_clauses
$\vdash \forall\ a$
$\bullet\ a \setminus \{\} = a$
$\quad \wedge \{\} \setminus a = \{\}$
$\quad \wedge a \setminus \mathit{Universe} = \{\}$
$\quad \wedge \mathit{Universe} \setminus a = \sim a$
$\quad \wedge a \setminus a = \{\}$

⊖_clauses $\vdash \forall\ a$
$\bullet\ a \ominus \{\} = a$
$\quad \wedge \{\} \ominus a = a$
$\quad \wedge a \ominus \mathit{Universe} = \sim a$
$\quad \wedge \mathit{Universe} \ominus a = \sim a$
$\quad \wedge a \ominus a = \{\}$

⊆_clauses $\vdash \forall\ a \bullet\ a \subseteq a \wedge \{\} \subseteq a \wedge a \subseteq \mathit{Universe}$

⊂_clauses $\vdash \forall\ a \bullet\ \neg\ a \subset a \wedge \neg\ a \subset \{\} \wedge \{\} \subset \mathit{Universe}$

⋃_clauses $\vdash \bigcup \{\} = \{\} \wedge \bigcup \mathit{Universe} = \mathit{Universe}$

⋂_clauses $\vdash \bigcap \{\} = \mathit{Universe} \wedge \bigcap \mathit{Universe} = \{\}$

ℙ_clauses $\vdash \forall\ a$
$\bullet\ \mathbb{P}\ \{\} = \{\{\}\}$
$\quad \wedge \mathbb{P}\ \mathit{Universe} = \mathit{Universe}$
$\quad \wedge a \in \mathbb{P}\ a$
$\quad \wedge \{\} \in \mathbb{P}\ a$

∅_clauses $\vdash \forall\ x\ a$
$\bullet\ \{x | F\} = \{\}$
$\quad \wedge \neg\ x \in \{\}$
$\quad \wedge \{\} \cup a = a$
$\quad \wedge a \cup \{\} = a$
$\quad \wedge \{\} \cap a = \{\}$
$\quad \wedge a \cap \{\} = \{\}$
$\quad \wedge a \setminus \{\} = a$
$\quad \wedge \{\} \setminus a = \{\}$
$\quad \wedge a \ominus \{\} = a$
$\quad \wedge \{\} \ominus a = a$
$\quad \wedge \{\} \subseteq a$
$\quad \wedge (a \subseteq \{\} \Leftrightarrow a = \{\})$
$\quad \wedge (\{\} \subset a \Leftrightarrow \neg\ a = \{\})$

$\land \neg\ a \subset \{\}$
$\land \neg\ x \in \bigcup\ \{\}$
$\land\ x \in \bigcap\ \{\}$
$\land\ \{\} \in \mathbb{P}\ a$
$\land\ \mathbb{P}\ \{\} = \{\{\}\}$

∈_in_clauses $\vdash (\forall\ x\ y\ a$
$\bullet\ (x \in \mathit{Universe} \Leftrightarrow T)$
$\land\ (x \in \{\} \Leftrightarrow F)$
$\land\ (x \in \mathit{Insert}\ y\ a \Leftrightarrow x = y \lor x \in a))$
$\land\ (\forall\ x\ a\ b$
$\bullet\ (x \in a \cup b \Leftrightarrow x \in a \lor x \in b)$
$\land\ (x \in a \cap b \Leftrightarrow x \in a \land x \in b)$
$\land\ (x \in a \setminus b \Leftrightarrow x \in a \land \neg\ x \in b)$
$\land\ (x \in a \ominus b \Leftrightarrow \neg\ (x \in a \Leftrightarrow x \in b)))$
$\land\ (\forall\ x\ a$
$\bullet\ (x \in \bigcup\ a \Leftrightarrow (\exists\ s\bullet\ x \in s \land s \in a))$
$\land\ (x \in \bigcap\ a \Leftrightarrow (\forall\ s\bullet\ s \in a \Rightarrow x \in s))$
$\land\ (\forall\ x\ a\bullet\ x \in \mathbb{P}\ a \Leftrightarrow x \subseteq a))$

sets_ext_clauses
$\vdash \forall\ a\ b$
$\bullet\ (a \subset b$
$\Leftrightarrow (\forall\ x\bullet\ x \in a \Rightarrow x \in b)$
$\land\ (\exists\ x\bullet\ \neg\ x \in a \land x \in b))$
$\land\ (a \subseteq b \Leftrightarrow (\forall\ x\bullet\ x \in a \Rightarrow x \in b))$
$\land\ (a = b \Leftrightarrow (\forall\ x\bullet\ x \in a \Leftrightarrow x \in b))$

9.1.20 The Theory set_thms

9.1.20.1 Parents

sets

9.1.20.2 Children

fun_rel_thms

9.1.20.3 Constants

Nest	$'a\ SET\ SET \to BOOL$
SubsetClosed	$'a\ SET\ SET \to BOOL$
NestClosed	$'a\ SET\ SET \to BOOL$
Maximal$_\subseteq$	$'a\ SET\ SET \to 'a\ SET \to BOOL$
Choose	$'a\ SET \to 'a$

9.1.20.4 Definitions

Nest $\vdash \forall\ u\bullet\ Nest\ u \Leftrightarrow (\forall\ a\ b\bullet\ a \in u \land b \in u \Rightarrow a \subseteq b \lor b \subseteq a)$

SubsetClosed $\vdash \forall\ u\bullet\ SubsetClosed\ u \Leftrightarrow (\forall\ a\ b\bullet\ a \in u \land b \subseteq a \Rightarrow b \in u)$

NestClosed $\vdash \forall\ u\bullet\ NestClosed\ u \Leftrightarrow (\forall\ v\bullet\ v \subseteq u \land Nest\ v \Rightarrow \bigcup v \in u)$

Maximal$_\subseteq$
$$\vdash \forall\ u\ a \bullet\ Maximal_\subseteq\ u\ a \Leftrightarrow a \in u \land (\forall\ b\bullet\ b \in u \land a \subseteq b \Rightarrow b = a)$$

Choose
$$\vdash ConstSpec\ (\lambda\ Choose'\bullet\ \forall\ a\bullet\ \neg\ a = \{\} \Leftrightarrow Choose'\ a \in a)\ Choose$$

9.1.20.5 Theorems

Choose_consistent
$$\vdash Consistent\ (\lambda\ Choose'\bullet\ \forall\ a\bullet\ \neg\ a = \{\} \Leftrightarrow Choose'\ a \in a)$$

zorn_thm1
$$\vdash \forall\ u \bullet\ \neg\ u = \{\} \land SubsetClosed\ u \land NestClosed\ u \Rightarrow (\exists\ a\bullet\ Maximal_\subseteq\ u\ a)$$

9.1.21 The Theory sum

9.1.21.1 Parents

combin

9.1.21.2 Children

hol

9.1.21.3 Constants

IsSumRep	$'a \times 'b \times BOOL \to BOOL$
IsR	$'a + 'b \to BOOL$
IsL	$'a + 'b \to BOOL$
OutR	$'a + 'b \to 'b$
OutL	$'a + 'b \to 'a$
InR	$'b \to 'a + 'b$
InL	$'a \to 'a + 'b$

9.1.21.4 Types

$'1 + '2$

9.1.21.5 Fixity

Right Infix 300:

$+$

9.1.21.6 Definitions

IsSumRep
is_sum_rep_def

$$
\begin{array}{l}
\vdash \exists\ inl\ inr\ outl\ outr\ isl\ isr \\
\quad \bullet \forall\ x1\ x2\ y1\ y2\ z \\
\quad\quad \bullet\ IsSumRep\ (inl\ x1) \\
\quad\quad\quad \wedge\ IsSumRep\ (inr\ y1) \\
\quad\quad\quad \wedge\ (IsSumRep\ z \\
\quad\quad\quad\quad \Rightarrow inl\ (outl\ z) = z \vee inr\ (outr\ z) = z) \\
\quad\quad\quad \wedge\ (inl\ x1 = inl\ x2 \Leftrightarrow x1 = x2) \\
\quad\quad\quad \wedge\ (inr\ y1 = inr\ y2 \Leftrightarrow y1 = y2) \\
\quad\quad\quad \wedge\ \neg\ inl\ x1 = inr\ y1 \\
\quad\quad\quad \wedge\ \neg\ inr\ y1 = inl\ x1 \\
\quad\quad\quad \wedge\ outl\ (inl\ x1) = x1 \\
\quad\quad\quad \wedge\ outr\ (inr\ y1) = y1 \\
\quad\quad\quad \wedge\ (IsSumRep\ z \Rightarrow (isl\ z \Leftrightarrow inl\ (outl\ z) = z)) \\
\quad\quad\quad \wedge\ (IsSumRep\ z \Rightarrow (isr\ z \Leftrightarrow inr\ (outr\ z) = z))
\end{array}
$$

$+$

sum_def $\vdash \exists\ f \bullet\ TypeDefn\ IsSumRep\ f$

InL

InR
OutL
OutR
IsL
IsR
sum_clauses $\vdash \forall\ x1\ x2\ y1\ y2\ z$
$\bullet\ (InL\ (OutL\ z) = z \lor InR\ (OutR\ z) = z)$
$\land\ (InL\ x1 = InL\ x2 \Leftrightarrow x1 = x2)$
$\land\ (InR\ y1 = InR\ y2 \Leftrightarrow y1 = y2)$
$\land\ \neg\ InL\ x1 = InR\ y1$
$\land\ \neg\ InR\ y1 = InL\ x1$
$\land\ OutL\ (InL\ x1) = x1$
$\land\ OutR\ (InR\ y1) = y1$
$\land\ (IsL\ z \Leftrightarrow InL\ (OutL\ z) = z)$
$\land\ (IsR\ z \Leftrightarrow InR\ (OutR\ z) = z)$

9.1.21.7 Theorems

sum_cases_thm
$\vdash \forall\ x\bullet\ (\exists\ y\bullet\ x = InL\ y) \lor (\exists\ z\bullet\ x = InR\ z)$
sum_fns_thm $\vdash \forall\ f\ g\bullet\ \exists_1\ h\bullet\ h\ o\ InL = f \land h\ o\ InR = g$

9.1.22 The Theory $\mathbb{R}$

9.1.22.1 Parents

orders dyadic

9.1.22.2 Constants

Is_$\mathbb{R}$_Rep	$(\mathbb{N} \times \mathbb{Z})\ \mathit{SET} \to \mathit{BOOL}$
$\$<_{\mathbf{R}}$	$\mathbb{R} \to \mathbb{R} \to \mathit{BOOL}$
$\$\leq_{\mathbf{R}}$	$\mathbb{R} \to \mathbb{R} \to \mathit{BOOL}$
$\$>_{\mathbf{R}}$	$\mathbb{R} \to \mathbb{R} \to \mathit{BOOL}$
$\$\geq_{\mathbf{R}}$	$\mathbb{R} \to \mathbb{R} \to \mathit{BOOL}$
Sup	$\mathbb{R}\ \mathit{SET} \to \mathbb{R}$
$\mathbf{1_R}$	$\mathbb{R}$
$\mathbf{0_R}$	$\mathbb{R}$
$\$+_{\mathbf{R}}$	$\mathbb{R} \to \mathbb{R} \to \mathbb{R}$
$\sim_{\mathbf{R}}$	$\mathbb{R} \to \mathbb{R}$
$\$-_{\mathbf{R}}$	$\mathbb{R} \to \mathbb{R} \to \mathbb{R}$
$\mathbb{N}\mathbb{R}$	$\mathbb{N} \to \mathbb{R}$
$\mathbf{Abs_R}$	$\mathbb{R} \to \mathbb{R}$
$\$*_{\mathbf{R}}$	$\mathbb{R} \to \mathbb{R} \to \mathbb{R}$
$\$/_{\mathbf{R}}$	$\mathbb{R} \to \mathbb{R} \to \mathbb{R}$
$\$/_{\mathbf{N}}$	$\mathbb{N} \to \mathbb{N} \to \mathbb{R}$
$\$^{-\mathbf{1}}$	$\mathbb{R} \to \mathbb{R}$
$\$\hat{}_{\mathbf{N}}$	$\mathbb{R} \to \mathbb{N} \to \mathbb{R}$
$\mathbb{Z}\mathbb{R}$	$\mathbb{Z} \to \mathbb{R}$
$\$\hat{}_{\mathbf{Z}}$	$\mathbb{R} \to \mathbb{Z} \to \mathbb{R}$
Float	$\mathbb{N} \to \mathbb{Z} \to \mathbb{Z} \to \mathbb{R}$
$\mathbf{Max_R}$	$\mathbb{R}\ \mathit{LIST} \to \mathbb{R}$
$\mathbf{Min_R}$	$\mathbb{R}\ \mathit{LIST} \to \mathbb{R}$

9.1.22.3 Aliases

$<$	$\$<_R : \mathbb{R} \to \mathbb{R} \to \mathit{BOOL}$
$\leq$	$\$\leq_R : \mathbb{R} \to \mathbb{R} \to \mathit{BOOL}$
$>$	$\$>_R : \mathbb{R} \to \mathbb{R} \to \mathit{BOOL}$
$\geq$	$\$\geq_R : \mathbb{R} \to \mathbb{R} \to \mathit{BOOL}$
$+$	$\$+_R : \mathbb{R} \to \mathbb{R} \to \mathbb{R}$
$\sim$	$\sim_R : \mathbb{R} \to \mathbb{R}$
$-$	$\$-_R : \mathbb{R} \to \mathbb{R} \to \mathbb{R}$
Abs	$\mathit{Abs}_R : \mathbb{R} \to \mathbb{R}$
$*$	$\$*_R : \mathbb{R} \to \mathbb{R} \to \mathbb{R}$
$/$	$\$/_R : \mathbb{R} \to \mathbb{R} \to \mathbb{R}$
$/$	$\$/_N : \mathbb{N} \to \mathbb{N} \to \mathbb{R}$
$\hat{}$	$\$\hat{}_N : \mathbb{R} \to \mathbb{N} \to \mathbb{R}$
$\hat{}$	$\$\hat{}_Z : \mathbb{R} \to \mathbb{Z} \to \mathbb{R}$

9.1.22.4 Types

$\mathbb{R}$

9.1.22.5 Fixity

Left Infix 305: $-_{\mathbf{R}}$

Left Infix 315: $/$ $/_{\mathbf{N}}$ $/_{\mathbf{R}}$

Right Infix 210: $<_{\mathbf{R}}$ $>_{\mathbf{R}}$ $\leq_{\mathbf{R}}$ $\geq_{\mathbf{R}}$

Right Infix 300: $+_{\mathbf{R}}$

Right Infix 310: $*_{\mathbf{R}}$

Right Infix 320: $\hat{}_{\mathbf{N}}$ $\hat{}_{\mathbf{Z}}$

Postfix 320: $^{-1}$

9.1.22.6 Definitions

Is_ℝ_Rep $\vdash \forall\ a\bullet\ Is\_\mathbb{R}\_Rep\ a \Leftrightarrow a \in Cuts\ (Universe,\ \$dy\_less)$

ℝ

ℝ_def $\vdash \exists\ f\bullet\ TypeDefn\ Is\_\mathbb{R}\_Rep\ f$

$<_{\mathbf{R}}$

$$
\begin{array}{l}
\vdash ConstSpec\\
\quad (\lambda\ <'_R\\
\quad\ \bullet\ StrictLinearOrder\ (Universe,\ <'_R)\\
\quad\quad \wedge\ UnboundedBelow\ (Universe,\ <'_R)\\
\quad\quad \wedge\ UnboundedAbove\ (Universe,\ <'_R)\\
\quad\quad \wedge\ Complete\ (Universe,\ <'_R)\\
\quad\quad \wedge\ (\exists\ \iota\\
\quad\quad \bullet\ (\forall\ a\ b\bullet\ <'_R\ (\iota\ a)\ (\iota\ b) \Leftrightarrow a\ dy\_less\ b)\\
\quad\quad\quad \wedge\ \{x|\exists\ a\bullet\ \iota\ a = x\}\\
\quad\quad\quad\ \ DenseIn\ (Universe,\ <'_R)))\\
\quad \$<
\end{array}
$$

$\leq_{\mathbf{R}}$ $\vdash \forall\ x\ y\bullet\ x \leq y \Leftrightarrow x < y \vee x = y$

$>_{\mathbf{R}}$ $\vdash \forall\ x\ y\bullet\ x > y \Leftrightarrow y < x$

$\geq_{\mathbf{R}}$ $\vdash \forall\ x\ y\bullet\ x \geq y \Leftrightarrow y \leq x$

Sup

$$
\begin{array}{l}
\vdash ConstSpec\\
\quad (\lambda\ Sup'\\
\quad\ \bullet\ \forall\ A\\
\quad\quad \bullet\ \neg\ A = \{\} \wedge (\exists\ b\bullet\ \forall\ x\bullet\ x \in A \Rightarrow x \leq b)\\
\quad\quad\quad \Rightarrow (\forall\ x\bullet\ x \in A \Rightarrow x \leq Sup'\ A)\\
\quad\quad\quad\quad \wedge\ (\forall\ b\\
\quad\quad\quad\quad \bullet\ (\forall\ x\bullet\ x \in A \Rightarrow x \leq b) \Rightarrow Sup'\ A \leq b))\\
\quad Sup
\end{array}
$$

$+_{\mathbf{R}}$

$\mathbf{0}_{\mathbf{R}}$

$\mathbf{1}_{\mathbf{R}}$

$$
\begin{array}{l}
\vdash ConstSpec\\
\quad (\lambda\ (+'_R,\ 0'_R,\ 1'_R)\\
\quad\ \bullet\ (\forall\ x\ y\ z\\
\quad\quad \bullet\ +'_R\ (+'_R\ x\ y)\ z = +'_R\ x\ (+'_R\ y\ z))\\
\quad\quad \wedge\ (\forall\ x\ y\bullet\ +'_R\ x\ y = +'_R\ y\ x)\\
\quad\quad \wedge\ (\forall\ x\bullet\ +'_R\ x\ 0'_R = x)
\end{array}
$$

$\land\ (\forall\ x\bullet\ \exists\ y\bullet\ +'_R\ x\ y = 0'_R)$
$\land\ (\forall\ x\ y\ z\bullet\ y < z \Rightarrow +'_R\ x\ y < +'_R\ x\ z)$
$\land\ 0'_R < 1'_R)$
$(\$+,\ 0_R,\ 1_R)$

$\sim_\mathbf{R}$ $\vdash ConstSpec\ (\lambda \sim'_R\bullet\ \forall\ x\bullet\ x + \sim'_R\ x = 0_R) \sim$

$-_\mathbf{R}$ $\vdash \forall\ x\ y\bullet\ x - y = x + \sim y$

$\mathbb{NR}$ $\vdash 0. = 0_R \land (\forall\ m\bullet\ \mathbb{NR}\ (m + 1) = \mathbb{NR}\ m + 1_R)$

$\mathbf{Abs_R}$ $\vdash \forall\ x\bullet\ Abs\ x = (if\ 0. \le x\ then\ x\ else \sim x)$

$*_\mathbf{R}$ $\vdash ConstSpec$
$(\lambda\ *'_R$
$\bullet\ (\forall\ x\ y\ z$
$\bullet\ *'_R\ (*'_R\ x\ y)\ z = *'_R\ x\ (*'_R\ y\ z))$
$\land\ (\forall\ x\bullet\ *'_R\ x\ 1. = x)$
$\land\ (\forall\ x\ y\ z$
$\bullet\ *'_R\ x\ (y + z) = *'_R\ x\ y + *'_R\ x\ z)$
$\land\ (\forall\ x\ y\bullet\ *'_R\ x\ y = *'_R\ y\ x)$
$\land\ (\forall\ x\ y\bullet\ 0. < x \land 0. < y \Rightarrow 0. < *'_R\ x\ y))$
$\$*$

$/_\mathbf{R}$ $\vdash ConstSpec$
$(\lambda\ /'_R$
$\bullet\ (\forall\ y\ z\bullet\ \neg\ z = 0. \Rightarrow /'_R\ (y * z)\ z = y)$
$\land\ (\forall\ x\ y\ z$
$\bullet\ \neg\ z = 0. \Rightarrow /'_R\ (x * y)\ z = x * /'_R\ y\ z))$
$\$/$

$/_\mathbf{N}$ $\vdash \forall\ m\ n\bullet\ m\ /\ n = \mathbb{NR}\ m\ /\ \mathbb{NR}\ n$

$^{-1}$ $\vdash \forall\ x\bullet\ x\ ^{-1} = 1.\ /\ x$

$\hat{}_\mathbf{N}$ $\vdash (\forall\ x\bullet\ x\ \hat{}\ 0 = 1.) \land (\forall\ x\ m\bullet\ x\ \hat{}\ (m + 1) = x * x\ \hat{}\ m)$

$\mathbb{ZR}$ $\vdash ConstSpec$
$(\lambda\ \$\text{"}\mathbb{ZR}'\text{"}$
$\bullet\ \$\text{"}\mathbb{ZR}'\text{"}\ (\mathbb{NZ}\ 0) = 0.$
$\land\ \$\text{"}\mathbb{ZR}'\text{"}\ (\mathbb{NZ}\ 1) = 1.$
$\land\ (\forall\ i\ j$
$\bullet\ \$\text{"}\mathbb{ZR}'\text{"}\ (i + j) = \$\text{"}\mathbb{ZR}'\text{"}\ i + \$\text{"}\mathbb{ZR}'\text{"}\ j))$
$\mathbb{ZR}$

$\hat{}_\mathbf{Z}$ $\vdash ConstSpec$
$(\lambda\ \hat{}'_Z$
$\bullet\ \forall\ x\ m$
$\bullet\ \hat{}'_Z\ x\ (\mathbb{NZ}\ m) = x\ \hat{}\ m$
$\land\ \hat{}'_Z\ x\ (\sim (\mathbb{NZ}\ (m + 1)))$
$= (x\ \hat{}\ (m + 1))\ ^{-1})$
$\$\hat{}$

Float $\vdash \forall\ m\ p\ e\bullet\ Float\ m\ p\ e = \mathbb{NR}\ m * 10.\ \hat{}\ (e + \sim p)$

$\mathbf{Max_R}$ $\vdash ConstSpec$
$(\lambda\ Max'_R$
$\bullet\ (\forall\ x\bullet\ Max'_R\ [x] = x)$
$\land\ (\forall\ x\ y\ L$
$\bullet\ Max'_R\ (Cons\ x\ (Cons\ y\ L))$
$= (if\ x < Max'_R\ (Cons\ y\ L)$
$then\ Max'_R\ (Cons\ y\ L)$
$else\ x)))$
Max_R

Min$_{\mathbf{R}}$ $\vdash$ *ConstSpec*
$(\lambda\ Min'_R$
$\bullet\ (\forall\ x\bullet\ Min'_R\ [x] = x)$
$\land\ (\forall\ x\ y\ L$
$\bullet\ Min'_R\ (Cons\ x\ (Cons\ y\ L))$
$= (if\ x > Min'_R\ (Cons\ y\ L)$
$then\ Min'_R\ (Cons\ y\ L)$
$else\ x)))$
Min_R

9.1.22.7 Theorems

dy_less_order_lemmas_thm
$\vdash StrictLinearOrder\ (Universe, \$dy\_less)$
$\land\ UnboundedBelow\ (Universe, \$dy\_less)$
$\land\ UnboundedAbove\ (Universe, \$dy\_less)$
$\land\ Universe\ DenseIn\ (Universe, \$dy\_less)$

is_ℝ_rep_consistent_thm
$\vdash \exists\ a\bullet\ Is\_\mathbb{R}\_Rep\ a$

$<_{\mathbf{R}}$_consistent
$\vdash Consistent$
$(\lambda <'_R$
$\bullet\ StrictLinearOrder\ (Universe, <'_R)$
$\land\ UnboundedBelow\ (Universe, <'_R)$
$\land\ UnboundedAbove\ (Universe, <'_R)$
$\land\ Complete\ (Universe, <'_R)$
$\land\ (\exists\ \iota$
$\bullet\ (\forall\ a\ b\bullet\ <'_R\ (\iota\ a)\ (\iota\ b) \Leftrightarrow a\ dy\_less\ b)$
$\land\ \{x | \exists\ a\bullet\ \iota\ a = x\}$
$DenseIn\ (Universe, <'_R)))$

ℝ_unbounded_below_thm
$\vdash \forall\ x\bullet\ \exists\ y\bullet\ y < x$

ℝ_unbounded_above_thm
$\vdash \forall\ x\bullet\ \exists\ y\bullet\ x < y$

ℝ_less_irrefl_thm
$\vdash \forall\ x\bullet\ \neg\ x < x$

ℝ_less_antisym_thm
$\vdash \forall\ x\ y\bullet\ \neg\ (x < y \land y < x)$

ℝ_less_trans_thm
$\vdash \forall\ x\ y\ z\bullet\ x < y \land y < z \Rightarrow x < z$

ℝ_less_cases_thm
$\vdash \forall\ x\ y\bullet\ x < y \lor x = y \lor y < x$

ℝ_≤_cases_thm
$\vdash \forall\ x\ y\bullet\ x \le y \lor y \le x$

ℝ_≤_less_cases_thm
$\vdash \forall\ x\ y\bullet\ x \le y \lor y < x$

ℝ_eq_≤_thm $\vdash \forall\ x\ y\bullet\ x = y \Leftrightarrow x \le y \land y \le x$

ℝ_≤_antisym_thm
$\vdash \forall\ x\ y\bullet\ x \le y \land y \le x \Rightarrow x = y$

ℝ_less_≤_trans_thm
$\vdash \forall\ x\ y\ z\bullet\ x < y \land y \le z \Rightarrow x < z$

ℝ_≤_less_trans_thm
$\vdash \forall\ x\ y\ z \bullet\ x \le y \land y < z \Rightarrow x < z$

ℝ_≤_refl_thm $\vdash \forall\ x \bullet\ x \le x$

ℝ_≤_trans_thm
$\vdash \forall\ x\ y\ z \bullet\ x \le y \land y \le z \Rightarrow x \le z$

ℝ_≤_¬_less_thm
$\vdash \forall\ x\ y \bullet\ x \le y \Leftrightarrow \neg\ y < x$

ℝ_¬_≤_less_thm
$\vdash \forall\ x\ y \bullet\ \neg\ x \le y \Leftrightarrow y < x$

ℝ_less_¬_eq_thm
$\vdash \forall\ x\ y \bullet\ x < y \Rightarrow \neg\ x = y$

ℝ_less_dense_thm
$\vdash \forall\ x\ y \bullet\ x < y \Rightarrow (\exists\ z \bullet\ x < z \land z < y)$

ℝ_complete_thm
$\vdash \forall\ A$
$\quad \bullet\ \neg\ A = \{\} \land (\exists\ b \bullet\ \forall\ x \bullet\ x \in A \Rightarrow x \le b)$
$\quad\quad \Rightarrow (\exists\ s$
$\quad\quad \bullet\ (\forall\ x \bullet\ x \in A \Rightarrow x \le s)$
$\quad\quad\quad \land (\forall\ b \bullet\ (\forall\ x \bullet\ x \in A \Rightarrow x \le b) \Rightarrow s \le b))$

Sup_consistent
$\vdash Consistent$
$\quad (\lambda\ Sup'$
$\quad\quad \bullet\ \forall\ A$
$\quad\quad\quad \bullet\ \neg\ A = \{\} \land (\exists\ b \bullet\ \forall\ x \bullet\ x \in A \Rightarrow x \le b)$
$\quad\quad\quad\quad \Rightarrow (\forall\ x \bullet\ x \in A \Rightarrow x \le Sup'\ A)$
$\quad\quad\quad\quad\quad \land (\forall\ b$
$\quad\quad\quad\quad\quad \bullet\ (\forall\ x \bullet\ x \in A \Rightarrow x \le b) \Rightarrow Sup'\ A \le b))$

ℝ_sup_thm $\vdash \forall\ A\ a$
$\quad \bullet\ \neg\ A = \{\} \land (\forall\ x \bullet\ x \in A \Rightarrow x \le a)$
$\quad\quad \Rightarrow (\forall\ x \bullet\ x \in A \Rightarrow x \le Sup\ A)$
$\quad\quad\quad \land (\forall\ b \bullet\ (\forall\ x \bullet\ x \in A \Rightarrow x \le b) \Rightarrow Sup\ A \le b)$

ℝ_less_sup_thm
$\vdash \forall\ A$
$\quad \bullet\ \neg\ A = \{\} \land (\exists\ a \bullet\ \forall\ x \bullet\ x \in A \Rightarrow x \le a)$
$\quad\quad \Rightarrow (\forall\ x \bullet\ x < Sup\ A \Leftrightarrow (\exists\ y \bullet\ y \in A \land x < y))$

ℝ_less_sup_bc_thm
$\vdash \forall\ A\ x$
$\quad \bullet\ \neg\ A = \{\}$
$\quad\quad\quad \land (\exists\ a \bullet\ \forall\ x \bullet\ x \in A \Rightarrow x \le a)$
$\quad\quad\quad \land (\exists\ y \bullet\ y \in A \land x < y)$
$\quad\quad \Rightarrow x < Sup\ A$

ℝ_≤_sup_thm $\vdash \forall\ A\ a$
$\quad \bullet\ \neg\ A = \{\} \land (\exists\ a \bullet\ \forall\ x \bullet\ x \in A \Rightarrow x \le a)$
$\quad\quad \Rightarrow (\forall\ x$
$\quad\quad \bullet\ x \le Sup\ A$
$\quad\quad\quad \Leftrightarrow (\forall\ y \bullet\ (\forall\ z \bullet\ z \in A \Rightarrow z \le y) \Rightarrow x \le y))$

ℝ_≤_sup_bc_thm
$\vdash \forall\ A\ a\ x$
$\quad \bullet\ \neg\ A = \{\}$
$\quad\quad\quad \land (\exists\ a \bullet\ \forall\ x \bullet\ x \in A \Rightarrow x \le a)$
$\quad\quad\quad \land (\forall\ y \bullet\ (\forall\ z \bullet\ z \in A \Rightarrow z \le y) \Rightarrow x \le y)$

$\Rightarrow x \leq Sup\ A$

ℝ_∈_≤_sup_bc_thm

$\vdash \forall A\ x \bullet x \in A \land (\exists a \bullet \forall x \bullet x \in A \Rightarrow x \leq a) \Rightarrow x \leq Sup\ A$

ℝ_⊆_sup_thm $\vdash \forall A\ B$

$\bullet \neg A = \{\}$
$\land (\exists a \bullet \forall x \bullet x \in A \Rightarrow x \leq a)$
$\land \neg B = \{\}$
$\land (\exists b \bullet \forall y \bullet y \in B \Rightarrow y \leq b)$
$\land A \subseteq B$
$\Rightarrow Sup\ A \leq Sup\ B$

ℝ_sup_≤_bc_thm

$\vdash \forall A\ a\ x$
$\bullet \neg A = \{\}$
$\land (\exists a \bullet \forall x \bullet x \in A \Rightarrow x \leq a)$
$\land (\forall y \bullet y \in A \Rightarrow y \leq x)$
$\Rightarrow Sup\ A \leq x$

ℝ_sup_less_bc_thm

$\vdash \forall A\ x\ z$
$\bullet \neg A = \{\}$
$\land (\exists a \bullet \forall x \bullet x \in A \Rightarrow x \leq a)$
$\land (\forall y \bullet y \in A \Rightarrow y \leq x)$
$\land x < z$
$\Rightarrow Sup\ A < z$

ℝ_sup_eq_bc_thm

$\vdash \forall A\ a\ s$
$\bullet \neg A = \{\}$
$\land (\forall x \bullet x \in A \Rightarrow x \leq s)$
$\land (\forall x \bullet (\forall y \bullet y \in A \Rightarrow y \leq x) \Rightarrow s \leq x)$
$\Rightarrow Sup\ A = s$

ℝ_eq_sup_bc_thm

$\vdash \forall A\ a\ s$
$\bullet \neg A = \{\}$
$\land (\forall x \bullet x \in A \Rightarrow x \leq s)$
$\land (\forall x \bullet (\forall y \bullet y \in A \Rightarrow y \leq x) \Rightarrow s \leq x)$
$\Rightarrow s = Sup\ A$

ℝ_less_sup_∈_thm

$\vdash \forall A\ a$
$\bullet \neg A = \{\} \land (\forall x \bullet x \in A \Rightarrow x \leq a) \land \neg Sup\ A \in A$
$\Rightarrow (\forall x$
$\bullet x < Sup\ A \Rightarrow (\exists y \bullet x < y \land y < Sup\ A \land y \in A))$

$+_R$_consistent
0_R_consistent
1_R_consistent

$\vdash Consistent$
$(\lambda (+'_R, 0'_R, 1'_R)$
$\bullet (\forall x\ y\ z$
$\bullet +'_R (+'_R\ x\ y)\ z = +'_R\ x\ (+'_R\ y\ z))$
$\land (\forall x\ y \bullet +'_R\ x\ y = +'_R\ y\ x)$
$\land (\forall x \bullet +'_R\ x\ 0'_R = x)$
$\land (\forall x \bullet \exists y \bullet +'_R\ x\ y = 0'_R)$
$\land (\forall x\ y\ z \bullet y < z \Rightarrow +'_R\ x\ y < +'_R\ x\ z)$

$\wedge\ 0'_R < 1'_R)$

$\sim_R$_consistent
$\vdash$ *Consistent* $(\lambda \sim'_R\bullet\ \forall\ x\bullet\ x + \sim'_R\ x = 0_R)$

ℝ_plus_assoc_thm
$\vdash \forall\ x\ y\ z\bullet\ (x + y) + z = x + y + z$

ℝ_plus_comm_thm
$\vdash \forall\ x\ y\bullet\ x + y = y + x$

ℝ_plus_unit_thm
$\vdash \forall\ x\bullet\ x + 0_R = x$

ℝ_plus_mono_thm
$\vdash \forall\ x\ y\ z\bullet\ y < z \Rightarrow x + y < x + z$

ℝ_plus_assoc_thm1
$\vdash \forall\ x\ y\ z\bullet\ x + y + z = (x + y) + z$

ℝ_plus_mono_thm1
$\vdash \forall\ x\ y\ z\bullet\ y < z \Rightarrow y + x < z + x$

ℝ_plus_mono_thm2
$\vdash \forall\ x\ y\ s\ t\bullet\ x < y \wedge s < t \Rightarrow x + s < y + t$

ℝ_plus_0_thm $\vdash \forall\ x\bullet\ x + 0. = x \wedge 0. + x = x$

ℝ_0_1_thm $\vdash 0_R = 0. \wedge 1_R = 1.$

ℝ_plus_order_thm
$\vdash \forall\ x\ y\ z$
$\bullet\ y + x = x + y$
$\wedge\ (x + y) + z = x + y + z$
$\wedge\ y + x + z = x + y + z$

ℝ_plus_minus_thm
$\vdash \forall\ x\bullet\ x + \sim x = 0. \wedge \sim x + x = 0.$

ℝ_eq_thm $\vdash \forall\ x\ y\bullet\ x = y \Leftrightarrow x + \sim y = 0.$

ℕℝ_plus_homomorphism_thm
$\vdash \forall\ m\ n\bullet\ \mathbb{NR}\ (m + n) = \mathbb{NR}\ m + \mathbb{NR}\ n$

ℝ_minus_clauses
$\vdash \forall\ x\ y$
$\bullet\ \sim (\sim x) = x$
$\wedge\ x + \sim x = 0.$
$\wedge\ \sim x + x = 0.$
$\wedge\ \sim (x + y) = \sim x + \sim y$
$\wedge\ \sim 0. = 0.$

ℝ_minus_eq_thm
$\vdash \forall\ x\ y\bullet\ \sim x = \sim y \Leftrightarrow x = y$

ℕℝ_0_less_thm
$\vdash \forall\ m\bullet\ 0. < \mathbb{NR}\ (m + 1)$

ℕℝ_one_one_thm
$\vdash \forall\ m\ n\bullet\ \mathbb{NR}\ m = \mathbb{NR}\ n \Leftrightarrow m = n$

ℝ_plus_clauses
$\vdash \forall\ x\ y\ z$
$\bullet\ (x + z = y + z \Leftrightarrow x = y)$
$\wedge\ (z + x = y + z \Leftrightarrow x = y)$
$\wedge\ (x + z = z + y \Leftrightarrow x = y)$
$\wedge\ (z + x = z + y \Leftrightarrow x = y)$
$\wedge\ (x + z = z \Leftrightarrow x = 0.)$
$\wedge\ (z + x = z \Leftrightarrow x = 0.)$
$\wedge\ (z = z + y \Leftrightarrow y = 0.)$

$$\begin{array}{l} \land (z = y + z \Leftrightarrow y = 0.) \\ \land x + 0. = x \\ \land 0. + x = x \\ \land \neg 1. = 0. \\ \land \neg 0. = 1. \end{array}$$

ℝ_less_less_0_thm

$\vdash \forall x\ y \bullet x < y \Leftrightarrow x + \sim y < 0.$

ℝ_less_clauses

$$\begin{array}{l} \vdash \forall x\ y\ z \\ \quad \bullet (x + z < y + z \Leftrightarrow x < y) \\ \qquad \land (z + x < y + z \Leftrightarrow x < y) \\ \qquad \land (x + z < z + y \Leftrightarrow x < y) \\ \qquad \land (z + x < z + y \Leftrightarrow x < y) \\ \qquad \land (x + z < z \Leftrightarrow x < 0.) \\ \qquad \land (z + x < z \Leftrightarrow x < 0.) \\ \qquad \land (x < z + x \Leftrightarrow 0. < z) \\ \qquad \land (x < x + z \Leftrightarrow 0. < z) \\ \qquad \land \neg x < x \\ \qquad \land 0. < 1. \\ \qquad \land \neg 1. < 0. \end{array}$$

ℝ_less_0_less_thm

$\vdash \forall x\ y \bullet x < y \Leftrightarrow 0. < y + \sim x$

ℝ_≤_clauses

$$\begin{array}{l} \vdash \forall x\ y\ z \\ \quad \bullet (x + z \le y + z \Leftrightarrow x \le y) \\ \qquad \land (z + x \le y + z \Leftrightarrow x \le y) \\ \qquad \land (x + z \le z + y \Leftrightarrow x \le y) \\ \qquad \land (z + x \le z + y \Leftrightarrow x \le y) \\ \qquad \land (x + z \le z \Leftrightarrow x \le 0.) \\ \qquad \land (z + x \le z \Leftrightarrow x \le 0.) \\ \qquad \land (x \le z + x \Leftrightarrow 0. \le z) \\ \qquad \land (x \le x + z \Leftrightarrow 0. \le z) \\ \qquad \land x \le x \\ \qquad \land 0. \le 1. \\ \qquad \land \neg 1. \le 0. \end{array}$$

ℝ_≤_≤_0_thm $\vdash \forall x\ y \bullet x \le y \Leftrightarrow x + \sim y \le 0.$

ℝ_≤_0_≤_thm $\vdash \forall x\ y \bullet x \le y \Leftrightarrow 0. \le y + \sim x$

ℕℝ_less_thm $\vdash \forall m\ n \bullet ℕℝ\ m < ℕℝ\ n \Leftrightarrow m < n$

ℝ_less_strong_dense_thm

$\vdash \forall x\ y \bullet x < y \Rightarrow (\exists d \bullet 0. < d \land x + d < y)$

ℕℝ_≤_thm $\vdash \forall m\ n \bullet ℕℝ\ m \le ℕℝ\ n \Leftrightarrow m \le n$

ℝ_sup_plus_thm

$$\begin{array}{l} \vdash \forall A\ a\ x \\ \quad \bullet \neg A = \{\} \land (\forall x \bullet x \in A \Rightarrow x \le a) \\ \qquad \Rightarrow Sup\ A + x = Sup\ \{t | \exists a \bullet a \in A \land t < a + x\} \end{array}$$

ℝ_sup_plus_sup_thm

$$\begin{array}{l} \vdash \forall A\ a\ B\ b \\ \quad \bullet \neg A = \{\} \\ \qquad \land (\forall x \bullet x \in A \Rightarrow x \le a) \\ \qquad \land \neg B = \{\} \\ \qquad \land (\forall y \bullet y \in B \Rightarrow y \le b) \\ \qquad \Rightarrow Sup\ A + Sup\ B \end{array}$$

$= Sup\ \{t | \exists\ a\ b \bullet\ a \in A \land b \in B \land t < a + b\}$

ℝ_delta_induction_thm

$$\begin{array}{l} \vdash \forall\ x\ p \\ \quad \bullet\ (\exists\ d \\ \qquad \bullet\ 0. < d \\ \qquad\quad \land (\exists\ e \\ \qquad\quad \bullet\ d < e \land (\forall\ t \bullet\ x < t \land t < x + e \Rightarrow p\ t)) \\ \qquad\quad \land (\forall\ s \bullet\ x < s \land p\ s \Rightarrow p\ (s + d))) \\ \qquad \Rightarrow (\forall\ y \bullet\ x < y \Rightarrow p\ y) \end{array}$$

ℝ_ord_pres_strict_thm

$$\begin{array}{l} \vdash \forall\ f \\ \quad \bullet\ (\forall\ x\ y \bullet\ x < y \Rightarrow f\ x < f\ y) \\ \qquad \Rightarrow (\forall\ x\ y \bullet\ f\ x < f\ y \Rightarrow x < y) \end{array}$$

ℝ_add_hom_0_thm

$$\begin{array}{l} \vdash \forall\ f \\ \quad \bullet\ (\forall\ x\ y \bullet\ f\ (x + y) = f\ x + f\ y) \Rightarrow (\forall\ x \bullet\ f\ 0. = 0.) \end{array}$$

ℝ_add_hom_minus_thm

$$\begin{array}{l} \vdash \forall\ f \\ \quad \bullet\ (\forall\ x\ y \bullet\ f\ (x + y) = f\ x + f\ y) \\ \qquad \Rightarrow (\forall\ x \bullet\ f\ (\sim x) = \sim (f\ x)) \end{array}$$

ℝ_add_hom_extension_thm

$$\begin{array}{l} \vdash \forall\ f \\ \quad \bullet\ (\forall\ x\ y \bullet\ 0. \le x \land 0. \le y \Rightarrow f\ (x + y) = f\ x + f\ y) \\ \qquad\ \land (\forall\ x \bullet\ 0. \le x \Rightarrow f\ (\sim x) = \sim (f\ x)) \\ \qquad \Rightarrow (\forall\ x\ y \bullet\ f\ (x + y) = f\ x + f\ y) \end{array}$$

ℝ_monoid_delta_dense_thm

$$\begin{array}{l} \vdash \forall\ G\ d \\ \quad \bullet\ 0. \in G \\ \qquad\ \land (\forall\ g\ h \bullet\ g \in G \land h \in G \Rightarrow g + h \in G) \\ \qquad\ \land d \in G \\ \qquad\ \land 0. < d \\ \qquad \Rightarrow (\forall\ x \\ \qquad \bullet\ 0. < x \Rightarrow (\exists\ g \bullet\ g \in G \land g \le x \land x < g + d)) \end{array}$$

ℝ_monoid_dense_thm

$$\begin{array}{l} \vdash \forall\ G \\ \quad \bullet\ 0. \in G \\ \qquad\ \land (\forall\ g\ h \bullet\ g \in G \land h \in G \Rightarrow g + h \in G) \\ \qquad\ \land (\forall\ x \bullet\ 0. < x \Rightarrow (\exists\ g \bullet\ g \in G \land 0. < g \land g < x)) \\ \qquad \Rightarrow (\forall\ x\ y \\ \qquad \bullet\ 0. < x \land x < y \Rightarrow (\exists\ g \bullet\ g \in G \land x < g \land g < y)) \end{array}$$

ℝ_subgroup_dense_thm

$$\begin{array}{l} \vdash \forall\ G \\ \quad \bullet\ 0. \in G \\ \qquad\ \land (\forall\ g\ h \bullet\ g \in G \land h \in G \Rightarrow g + h \in G) \\ \qquad\ \land (\forall\ g \bullet\ g \in G \Rightarrow \sim g \in G) \\ \qquad\ \land (\forall\ x \bullet\ 0. < x \Rightarrow (\exists\ g \bullet\ g \in G \land 0. < g \land g < x)) \\ \qquad \Rightarrow (\forall\ x\ y \bullet\ x < y \Rightarrow (\exists\ g \bullet\ g \in G \land x < g \land g < y)) \end{array}$$

ℝ_semigroup_dense_thm

$$\begin{array}{l} \vdash \forall\ G \\ \quad \bullet\ (\forall\ g\ h \bullet\ g \in G \land h \in G \Rightarrow g + h \in G) \\ \qquad\ \land (\forall\ x \bullet\ 0. < x \Rightarrow (\exists\ g \bullet\ g \in G \land 0. < g \land g < x)) \end{array}$$

$\Rightarrow (\forall\ x\ y$
$\bullet\ 0. < x \wedge x < y \Rightarrow (\exists\ g\bullet\ g \in G \wedge x < g \wedge g < y))$

ℝ_add_hom_image_group_thm

$\vdash \forall\ f\ I$
$\bullet\ (\forall\ x\ y\bullet\ f\ (x + y) = f\ x + f\ y)$
$\wedge\ I = \{b|\exists\ a\bullet\ b = f\ a\}$
$\Rightarrow 0. \in I$
$\wedge\ (\forall\ g\ h\bullet\ g \in I \wedge h \in I \Rightarrow g + h \in I)$
$\wedge\ (\forall\ g\bullet\ g \in I \Rightarrow \sim g \in I)$

ℝ_add_hom_kernel_group_thm

$\vdash \forall\ f\ K$
$\bullet\ (\forall\ x\ y\bullet\ f\ (x + y) = f\ x + f\ y) \wedge K = \{a|f\ a = 0.\}$
$\Rightarrow 0. \in K$
$\wedge\ (\forall\ g\ h\bullet\ g \in K \wedge h \in K \Rightarrow g + h \in K)$
$\wedge\ (\forall\ g\bullet\ g \in K \Rightarrow \sim g \in K)$

ℝ_opah_thm

$\vdash \forall\ f$
$\bullet\ (\forall\ x\ y\bullet\ f\ (x + y) = f\ x + f\ y)$
$\wedge\ (\forall\ x\bullet\ 0. < x \Rightarrow 0. < f\ x)$
$\Rightarrow (\forall\ x\ y\bullet\ x < y \Rightarrow f\ x < f\ y)$

ℝ_opah_strict_thm

$\vdash \forall\ f$
$\bullet\ (\forall\ x\ y\bullet\ f\ (x + y) = f\ x + f\ y)$
$\wedge\ (\forall\ x\bullet\ 0. < x \Rightarrow 0. < f\ x)$
$\Rightarrow (\forall\ x\ y\bullet\ f\ x < f\ y \Rightarrow x < y)$

ℝ_opah_one_one_thm

$\vdash \forall\ f$
$\bullet\ (\forall\ x\ y\bullet\ f\ (x + y) = f\ x + f\ y)$
$\wedge\ (\forall\ x\bullet\ 0. < x \Rightarrow 0. < f\ x)$
$\Rightarrow OneOne\ f$

ℝ_opah_dense_image_thm

$\vdash \forall\ f\ e$
$\bullet\ (\forall\ x\ y\bullet\ f\ (x + y) = f\ x + f\ y)$
$\wedge\ (\forall\ x\bullet\ 0. < x \Rightarrow 0. < f\ x)$
$\wedge\ 0. < e$
$\Rightarrow (\exists\ d\bullet\ 0. < d \wedge f\ d < e)$

ℝ_opah_onto_thm

$\vdash \forall\ f$
$\bullet\ (\forall\ x\ y\bullet\ f\ (x + y) = f\ x + f\ y)$
$\wedge\ (\forall\ x\bullet\ 0. < x \Rightarrow 0. < f\ x)$
$\Rightarrow Onto\ f$

ℝ_opah_inverse_add_hom_thm

$\vdash \forall\ f\ g$
$\bullet\ (\forall\ x\ y\bullet\ f\ (x + y) = f\ x + f\ y)$
$\wedge\ (\forall\ x\bullet\ 0. < x \Rightarrow 0. < f\ x)$
$\wedge\ (\forall\ x\bullet\ g\ (f\ x) = x)$
$\wedge\ (\forall\ x\bullet\ f\ (g\ x) = x)$
$\Rightarrow (\forall\ x\ y\bullet\ g\ (x + y) = g\ x + g\ y)$
$\wedge\ (\forall\ x\bullet\ 0. < x \Rightarrow 0. < g\ x)$

ℝ_opah_inverse_thm

$\vdash \forall\ f$
$\bullet\ (\forall\ x\ y\bullet\ f\ (x + y) = f\ x + f\ y)$

$$\begin{array}{l}
\qquad\quad \land\ (\forall\ x\bullet\ 0. < x \Rightarrow 0. < f\ x) \\
\qquad \Rightarrow (\exists\ g \\
\qquad \bullet\ (\forall\ x\bullet\ g\ (f\ x) = x) \\
\qquad\qquad \land\ (\forall\ x\bullet\ f\ (g\ x) = x) \\
\qquad\qquad \land\ (\forall\ x\ y\bullet\ g\ (x + y) = g\ x + g\ y) \\
\qquad\qquad \land\ (\forall\ x\bullet\ 0. < x \Rightarrow 0. < g\ x))
\end{array}$$

ℝ_copah_id_thm

$$\begin{array}{l}
\vdash \exists\ \iota \\
\quad \bullet\ (\forall\ x\bullet\ \iota\ x = x) \\
\qquad \land\ (\forall\ x\ y\bullet\ \iota\ (x + y) = \iota\ x + \iota\ y) \\
\qquad \land\ (\forall\ x\bullet\ 0. < x \Rightarrow 0. < \iota\ x) \\
\qquad \land\ (\forall\ f \\
\qquad \bullet\ (\forall\ x\ y\bullet\ f\ (x + y) = f\ x + f\ y) \\
\qquad\qquad \Rightarrow (\forall\ x\bullet\ \iota\ (f\ x) = f\ (\iota\ x)))
\end{array}$$

ℝ_copah_double_thm

$$\begin{array}{l}
\vdash \exists\ \alpha \\
\quad \bullet\ (\forall\ x\bullet\ \alpha\ x = x + x) \\
\qquad \land\ (\forall\ x\ y\bullet\ \alpha\ (x + y) = \alpha\ x + \alpha\ y) \\
\qquad \land\ (\forall\ x\bullet\ 0. < x \Rightarrow 0. < \alpha\ x) \\
\qquad \land\ (\forall\ x\bullet\ 0. < x \Rightarrow x < \alpha\ x) \\
\qquad \land\ (\forall\ f \\
\qquad \bullet\ (\forall\ x\ y\bullet\ f\ (x + y) = f\ x + f\ y) \\
\qquad\qquad \Rightarrow (\forall\ x\bullet\ \alpha\ (f\ x) = f\ (\alpha\ x)))
\end{array}$$

ℝ_copah_halve_thm

$$\begin{array}{l}
\vdash \exists\ \beta \\
\quad \bullet\ (\forall\ x\bullet\ \beta\ x + \beta\ x = x) \\
\qquad \land\ (\forall\ x\ y\bullet\ \beta\ (x + y) = \beta\ x + \beta\ y) \\
\qquad \land\ (\forall\ x\bullet\ 0. < x \Rightarrow 0. < \beta\ x) \\
\qquad \land\ (\forall\ x\bullet\ 0. < x \Rightarrow \beta\ x < x) \\
\qquad \land\ (\forall\ f \\
\qquad \bullet\ (\forall\ x\ y\bullet\ f\ (x + y) = f\ x + f\ y) \\
\qquad\qquad \Rightarrow (\forall\ x\bullet\ \beta\ (f\ x) = f\ (\beta\ x)))
\end{array}$$

ℝ_copah_comp_thm

$$\begin{array}{l}
\vdash \forall\ \alpha\ \beta \\
\quad \bullet\ (\forall\ x\ y\bullet\ \alpha\ (x + y) = \alpha\ x + \alpha\ y) \\
\qquad\quad \land\ (\forall\ x\bullet\ 0. < x \Rightarrow 0. < \alpha\ x) \\
\qquad\quad \land\ (\forall\ f \\
\qquad\quad \bullet\ (\forall\ x\ y\bullet\ f\ (x + y) = f\ x + f\ y) \\
\qquad\qquad\quad \Rightarrow (\forall\ x\bullet\ \alpha\ (f\ x) = f\ (\alpha\ x))) \\
\qquad\quad \land\ (\forall\ x\ y\bullet\ \beta\ (x + y) = \beta\ x + \beta\ y) \\
\qquad\quad \land\ (\forall\ x\bullet\ 0. < x \Rightarrow 0. < \beta\ x) \\
\qquad\quad \land\ (\forall\ f \\
\qquad\quad \bullet\ (\forall\ x\ y\bullet\ f\ (x + y) = f\ x + f\ y) \\
\qquad\qquad\quad \Rightarrow (\forall\ x\bullet\ \beta\ (f\ x) = f\ (\beta\ x))) \\
\qquad \Rightarrow (\exists\ \gamma \\
\qquad \bullet\ (\forall\ x\bullet\ \gamma\ x = \alpha\ (\beta\ x)) \\
\qquad\qquad \land\ (\forall\ x\bullet\ 0. < x \Rightarrow 0. < \gamma\ x) \\
\qquad\qquad \land\ (\forall\ x\ y\bullet\ \gamma\ (x + y) = \gamma\ x + \gamma\ y) \\
\qquad\qquad \land\ (\forall\ f \\
\qquad\qquad \bullet\ (\forall\ x\ y\bullet\ f\ (x + y) = f\ x + f\ y) \\
\qquad\qquad\qquad \Rightarrow (\forall\ x\bullet\ \gamma\ (f\ x) = f\ (\gamma\ x))))
\end{array}$$

ℝ_copah_sum_thm

$$\begin{array}{l}
\vdash \forall\ \alpha\ \beta \\
\quad \bullet\ (\forall\ x\ y \bullet\ \alpha\ (x + y) = \alpha\ x + \alpha\ y) \\
\qquad \land\ (\forall\ x \bullet\ 0. < x \Rightarrow 0. < \alpha\ x) \\
\qquad \land\ (\forall\ f \\
\qquad \bullet\ (\forall\ x\ y \bullet\ f\ (x + y) = f\ x + f\ y) \\
\qquad\quad \Rightarrow (\forall\ x \bullet\ \alpha\ (f\ x) = f\ (\alpha\ x))) \\
\qquad \land\ (\forall\ x\ y \bullet\ \beta\ (x + y) = \beta\ x + \beta\ y) \\
\qquad \land\ (\forall\ x \bullet\ 0. < x \Rightarrow 0. < \beta\ x) \\
\qquad \land\ (\forall\ f \\
\qquad \bullet\ (\forall\ x\ y \bullet\ f\ (x + y) = f\ x + f\ y) \\
\qquad\quad \Rightarrow (\forall\ x \bullet\ \beta\ (f\ x) = f\ (\beta\ x))) \\
\quad\ \Rightarrow (\exists\ \gamma \\
\quad\ \bullet\ (\forall\ x \bullet\ \gamma\ x = \alpha\ x + \beta\ x) \\
\qquad\ \land\ (\forall\ x \bullet\ 0. < x \Rightarrow 0. < \gamma\ x) \\
\qquad\ \land\ (\forall\ x\ y \bullet\ \gamma\ (x + y) = \gamma\ x + \gamma\ y) \\
\qquad\ \land\ (\forall\ f \\
\qquad\ \bullet\ (\forall\ x\ y \bullet\ f\ (x + y) = f\ x + f\ y) \\
\qquad\qquad \Rightarrow (\forall\ x \bullet\ \gamma\ (f\ x) = f\ (\gamma\ x))))
\end{array}$$

ℝ_halve_closed_dense_thm

$$\begin{array}{l}
\vdash \forall\ A\ e \\
\quad \bullet\ 0. < e \\
\qquad \land\ e \in A \\
\qquad \land\ (\forall\ y \bullet\ y \in A \Rightarrow (\exists\ z \bullet\ z \in A \land z + z = y)) \\
\quad\ \Rightarrow (\forall\ d \bullet\ 0. < d \Rightarrow (\exists\ a \bullet\ a \in A \land 0. < a \land a < d))
\end{array}$$

ℝ_copah_dense_thm

$$\begin{array}{l}
\vdash \forall\ d\ x\ y \\
\quad \bullet\ 0. < d \land 0. < x \land x < y \\
\quad\ \Rightarrow (\exists\ \gamma \\
\quad\ \bullet\ (\forall\ x\ y \bullet\ \gamma\ (x + y) = \gamma\ x + \gamma\ y) \\
\qquad\ \land\ (\forall\ x \bullet\ 0. < x \Rightarrow 0. < \gamma\ x) \\
\qquad\ \land\ (\forall\ f \\
\qquad\ \bullet\ (\forall\ x\ y \bullet\ f\ (x + y) = f\ x + f\ y) \\
\qquad\qquad \Rightarrow (\forall\ x \bullet\ \gamma\ (f\ x) = f\ (\gamma\ x))) \\
\qquad\ \land\ x < \gamma\ d \\
\qquad\ \land\ \gamma\ d < y)
\end{array}$$

ℝ_opah_extension_thm1

$$\begin{array}{l}
\vdash \forall\ f \\
\quad \bullet\ (\forall\ x\ y \bullet\ 0. < x \land 0. < y \Rightarrow f\ (x + y) = f\ x + f\ y) \\
\qquad \land\ (\forall\ x \bullet\ 0. < x \Rightarrow 0. < f\ x) \\
\quad\ \Rightarrow (\exists\ \phi \\
\quad\ \bullet\ \phi\ 0. = 0. \\
\qquad\ \land\ (\forall\ x \bullet\ 0. < x \Rightarrow \phi\ x = f\ x) \\
\qquad\ \land\ (\forall\ x\ y \\
\qquad\ \bullet\ 0. \le x \land 0. \le y \Rightarrow \phi\ (x + y) = \phi\ x + \phi\ y))
\end{array}$$

ℝ_opah_extension_thm2

$$\begin{array}{l}
\vdash \forall\ f \\
\quad \bullet\ f\ 0. = 0. \\
\qquad \land\ (\forall\ x\ y \\
\qquad \bullet\ 0. \le x \land 0. \le y \Rightarrow f\ (x + y) = f\ x + f\ y) \\
\quad\ \Rightarrow (\exists\ \psi
\end{array}$$

$\bullet\ (\forall\ x\bullet\ \mathit{0.} \leq x \Rightarrow \psi\ x = f\ x)$
$\quad \wedge\ (\forall\ x\ y\bullet\ \psi\ (x + y) = \psi\ x + \psi\ y))$

ℝ_opah_extension_thm

$\vdash \forall\ f$
$\bullet\ (\forall\ x\ y\bullet\ \mathit{0.} < x \wedge \mathit{0.} < y \Rightarrow f\ (x + y) = f\ x + f\ y)$
$\quad \wedge\ (\forall\ x\bullet\ \mathit{0.} < x \Rightarrow \mathit{0.} < f\ x)$
$\quad \Rightarrow (\exists\ \phi$
$\quad \bullet\ (\forall\ x\bullet\ \mathit{0.} < x \Rightarrow \phi\ x = f\ x)$
$\quad\quad \wedge\ (\forall\ x\ y\bullet\ \phi\ (x + y) = \phi\ x + \phi\ y)$
$\quad\quad \wedge\ (\forall\ x\bullet\ \mathit{0.} < x \Rightarrow \mathit{0.} < \phi\ x))$

ℝ_opah_order_thm

$\vdash \forall\ f\ g\ d$
$\bullet\ (\forall\ x\ y\bullet\ f\ (x + y) = f\ x + f\ y)$
$\quad \wedge\ (\forall\ x\bullet\ \mathit{0.} < x \Rightarrow \mathit{0.} < f\ x)$
$\quad \wedge\ (\forall\ x\ y\bullet\ g\ (x + y) = g\ x + g\ y)$
$\quad \wedge\ (\forall\ x\bullet\ \mathit{0.} < x \Rightarrow \mathit{0.} < g\ x)$
$\quad \wedge\ \mathit{0.} < d$
$\quad \wedge\ f\ d < g\ d$
$\quad \Rightarrow (\forall\ x\bullet\ \mathit{0.} < x \Rightarrow f\ x < g\ x)$

ℝ_opah_eq_thm

$\vdash \forall\ f\ g\ d$
$\bullet\ (\forall\ x\ y\bullet\ f\ (x + y) = f\ x + f\ y)$
$\quad \wedge\ (\forall\ x\bullet\ \mathit{0.} < x \Rightarrow \mathit{0.} < f\ x)$
$\quad \wedge\ (\forall\ x\ y\bullet\ g\ (x + y) = g\ x + g\ y)$
$\quad \wedge\ (\forall\ x\bullet\ \mathit{0.} < x \Rightarrow \mathit{0.} < g\ x)$
$\quad \wedge\ \mathit{0.} < d$
$\quad \wedge\ f\ d = g\ d$
$\quad \Rightarrow f = g$

ℝ_opah_complete_thm

$\vdash \forall\ d\ e$
$\bullet\ \mathit{0.} < d \wedge \mathit{0.} < e$
$\quad \Rightarrow (\exists\ f$
$\quad \bullet\ (\forall\ x\ y\bullet\ f\ (x + y) = f\ x + f\ y)$
$\quad\quad \wedge\ (\forall\ x\bullet\ \mathit{0.} < x \Rightarrow \mathit{0.} < f\ x)$
$\quad\quad \wedge\ f\ d = e)$

$*_{\mathbf{R}}$_consistent

$\vdash$ *Consistent*
$(\lambda\ *'_R$
$\bullet\ (\forall\ x\ y\ z$
$\quad \bullet\ *'_R\ (*'_R\ x\ y)\ z = *'_R\ x\ (*'_R\ y\ z))$
$\quad \wedge\ (\forall\ x\bullet\ *'_R\ x\ \mathit{1.} = x)$
$\quad \wedge\ (\forall\ x\ y\ z$
$\quad \bullet\ *'_R\ x\ (y + z) = *'_R\ x\ y + *'_R\ x\ z)$
$\quad \wedge\ (\forall\ x\ y\bullet\ *'_R\ x\ y = *'_R\ y\ x)$
$\quad \wedge\ (\forall\ x\ y\bullet\ \mathit{0.} < x \wedge \mathit{0.} < y \Rightarrow \mathit{0.} < *'_R\ x\ y))$

ℝ_times_assoc_thm

$\vdash \forall\ x\ y\ z\bullet\ (x * y) * z = x * y * z$

ℝ_times_comm_thm

$\vdash \forall\ x\ y\bullet\ x * y = y * x$

ℝ_times_unit_thm

$\vdash \forall\ x\bullet\ x * \mathit{1.} = x$

ℝ_0_less_0_less_times_thm
$\vdash \forall\ x\ y \bullet\ 0. < x \land 0. < y \Rightarrow 0. < x * y$

ℝ_times_assoc_thm1
$\vdash \forall\ x\ y\ z \bullet\ x * y * z = (x * y) * z$

ℝ_times_plus_distrib_thm
$\vdash \forall\ x\ y\ z$
$\bullet\ x * (y + z) = x * y + x * z$
$\land\ (x + y) * z = x * z + y * z$

ℝ_times_order_thm
$\vdash \forall\ x\ y\ z$
$\bullet\ y * x = x * y$
$\land\ (x * y) * z = x * y * z$
$\land\ y * x * z = x * y * z$

ℝ_times_0_thm
$\vdash \forall\ x \bullet\ x * 0. = 0. \land 0. * x = 0.$

ℝ_times_1_thm
$\vdash \forall\ x \bullet\ x * 1. = x \land 1. * x = x$

ℕℝ_times_homomorphism_thm
$\vdash \forall\ m\ n \bullet\ \mathbb{NR}\ (m * n) = \mathbb{NR}\ m * \mathbb{NR}\ n$

ℝ_times_minus_thm
$\vdash \forall\ x\ y$
$\bullet\ \sim x * y = \sim (x * y)$
$\land\ x * \sim y = \sim (x * y)$
$\land\ \sim x * \sim y = x * y$

$/_{\mathbf{R}}$_consistent
$\vdash$ *Consistent*
$(\lambda\ /'_R$
$\bullet\ (\forall\ y\ z \bullet\ \lnot\ z = 0. \Rightarrow /'_R\ (y * z)\ z = y)$
$\land\ (\forall\ x\ y\ z$
$\bullet\ \lnot\ z = 0. \Rightarrow /'_R\ (x * y)\ z = x * /'_R\ y\ z))$

ℝ_over_times_recip_thm
$\vdash \forall\ z \bullet\ \lnot\ z = 0. \Rightarrow (\forall\ x \bullet\ x\ /\ z = x * z^{\ -1})$

ℝ_times_recip_thm
$\vdash \forall\ z \bullet\ \lnot\ z = 0. \Rightarrow z * z^{\ -1} = 1.$

ℝ_eq_recip_thm
$\vdash \forall\ z \bullet\ \lnot\ z = 0. \Rightarrow (\forall\ y \bullet\ y = z \Leftrightarrow y * z^{\ -1} = 1.)$

ℝ_times_cancel_thm
$\vdash \forall\ x\ y\ z \bullet\ \lnot\ z = 0. \Rightarrow (x * z = y * z \Leftrightarrow x = y)$

ℝ_times_eq_0_thm
$\vdash \forall\ x\ y \bullet\ x * y = 0. \Leftrightarrow x = 0. \lor y = 0.$

ℝ_times_clauses
$\vdash \forall\ x$
$\bullet\ 0. * x = 0. \land x * 0. = 0. \land x * 1. = x \land 1. * x = x$

ℝ_times_mono_⇔_thm
$\vdash \forall\ x \bullet\ 0. < x \Rightarrow (\forall\ y\ z \bullet\ y < z \Leftrightarrow x * y < x * z)$

ℝ_times_mono_thm
$\vdash \forall\ x\ y\ z \bullet\ 0. < x \land y < z \Rightarrow x * y < x * z$

ℝ_0_≤_0_≤_times_thm
$\vdash \forall\ x\ y \bullet\ 0. \le x \land 0. \le y \Rightarrow 0. \le x * y$

ℝ_¬_recip_0_thm
$\vdash \forall\ z \bullet\ \lnot\ z = 0. \Rightarrow \lnot\ z^{\ -1} = 0.$

ℝ_recip_clauses
$\vdash (1.^{-1} = 1.$
$\quad \wedge (\forall w$
$\quad \bullet \neg w = 0.$
$\quad\quad \Rightarrow w^{-1\ -1} = w$
$\quad\quad\quad \wedge w * w^{-1} = 1.$
$\quad\quad\quad \wedge w^{-1} * w = 1.))$
$\quad \wedge (\forall w\ z$
$\quad \bullet \neg w = 0. \wedge \neg z = 0.$
$\quad\quad \Rightarrow (w * z)^{-1} = w^{-1} * z^{-1})$

ℝ_cross_mult_eq_thm
$\vdash \forall w\ z$
$\quad \bullet \neg w = 0. \wedge \neg z = 0.$
$\quad\quad \Rightarrow (\forall x\ y \bullet x / w = y / z \Leftrightarrow x * z = w * y)$

ℝ_less_¬_eq_0_thm
$\vdash \forall z \bullet 0. < z \Rightarrow \neg z = 0.$

ℝ_0_less_0_less_recip_thm
$\vdash \forall z \bullet 0. < z \Rightarrow 0. < z^{-1}$

ℝ_cross_mult_less_thm
$\vdash \forall w\ z$
$\quad \bullet 0. < w \wedge 0. < z$
$\quad\quad \Rightarrow (\forall x\ y \bullet x / w < y / z \Leftrightarrow x * z < w * y)$

ℝ_over_cancel_eq_thm
$\vdash \forall w\ z$
$\quad \bullet \neg w = 0. \wedge \neg z = 0.$
$\quad\quad \Rightarrow (\forall x \bullet (x * z) / (w * z) = x / w)$

ℝ_over_plus_over_thm
$\vdash \forall x\ y\ u\ v$
$\quad \bullet \neg u = 0. \wedge \neg v = 0.$
$\quad\quad \Rightarrow x / u + y / v = (x * v + y * u) / (u * v)$

ℝ_0_over_thm $\vdash \forall z \bullet \neg z = 0. \Rightarrow 0. / z = 0.$

ℝ_over_1_thm $\vdash \forall x \bullet x / 1. = x$

ℕℝ_plus_homomorphism_thm1
$\vdash \forall m\ n \bullet \mathbb{N}\mathbb{R}\ m + \mathbb{N}\mathbb{R}\ n = \mathbb{N}\mathbb{R}\ (m + n)$

ℕℝ_times_homomorphism_thm1
$\vdash \forall m\ n \bullet \mathbb{N}\mathbb{R}\ m * \mathbb{N}\mathbb{R}\ n = \mathbb{N}\mathbb{R}\ (m * n)$

ℝ_frac_cross_mult_eq_thm
$\vdash \forall m\ n\ i\ j$
$\quad \bullet i / (m + 1) = j / (n + 1)$
$\quad\quad \Leftrightarrow i * (n + 1) = j * (m + 1)$

ℝ_frac_cancel_eq_thm
$\vdash \forall i\ m\ n$
$\quad \bullet (i * (n + 1)) / ((m + 1) * (n + 1)) = i / (m + 1)$

ℝ_frac_0_thm $\vdash \forall m \bullet 0 / (m + 1) = 0.$

ℝ_frac_ℕ_thm $\vdash \forall i \bullet i / 1 = \mathbb{N}\mathbb{R}\ i$

ℝ_frac_plus_frac_thm
$\vdash \forall i\ j\ k\ m\ n$
$\quad \bullet i / (m + 1) + j / (n + 1)$
$\quad\quad = (i * (n + 1) + j * (m + 1))$
$\quad\quad\quad / ((m + 1) * (n + 1))$

ℝ_frac_minus_frac_thm

$\vdash \forall\ i\ j\ k\ m\ n$
$\quad \bullet\ j * (m + 1) \leq i * (n + 1)$
$\qquad \Rightarrow i\ /\ (m + 1) + \sim (j\ /\ (n + 1))$
$\qquad\quad = (i * (n + 1) - j * (m + 1))$
$\qquad\qquad /\ ((m + 1) * (n + 1))$

ℝ_frac_minus_frac_thm1

$\vdash \forall\ i\ j\ m\ n$
$\quad \bullet\ i * (n + 1) \leq j * (m + 1)$
$\qquad \Rightarrow i\ /\ (m + 1) + \sim (j\ /\ (n + 1))$
$\qquad\quad = \sim$
$\qquad\qquad ((j * (m + 1) - i * (n + 1))$
$\qquad\qquad\quad /\ ((m + 1) * (n + 1)))$

ℝ_over_times_over_thm

$\vdash \forall\ x\ y\ u\ v$
$\quad \bullet\ \neg\ u = 0. \wedge \neg\ v = 0.$
$\qquad \Rightarrow x\ /\ u * y\ /\ v = (x * y)\ /\ (u * v)$

ℝ_frac_times_frac_thm

$\vdash \forall\ i\ j\ m\ n$
$\quad \bullet\ i\ /\ (m + 1) * j\ /\ (n + 1)$
$\qquad = (i * j)\ /\ ((m + 1) * (n + 1))$

ℝ_over_recip_thm

$\vdash \forall\ u\ v \bullet\ \neg\ u = 0. \wedge \neg\ v = 0. \Rightarrow (u\ /\ v)\ ^{-1} = v\ /\ u$

ℝ_frac_recip_thm

$\vdash \forall\ m\ n \bullet\ ((m + 1)\ /\ (n + 1))\ ^{-1} = (n + 1)\ /\ (m + 1)$

ℝ_minus_recip_thm

$\vdash \forall\ z \bullet\ \neg\ z = 0. \Rightarrow \sim z\ ^{-1} = \sim (z\ ^{-1})$

ℝ_over_eq_0_thm

$\vdash \forall\ u\ v \bullet\ \neg\ u = 0. \wedge \neg\ v = 0. \Rightarrow \neg\ u\ /\ v = 0.$

ℝ_over_over_over_thm

$\vdash \forall\ x\ y\ u\ v$
$\quad \bullet\ \neg\ u = 0. \wedge \neg\ v = 0. \wedge \neg\ y = 0.$
$\qquad \Rightarrow x\ /\ u\ /\ (y\ /\ v) = (x * v)\ /\ (u * y)$

ℝ_frac_less_frac_thm

$\vdash \forall\ i\ j\ m\ n$
$\quad \bullet\ i\ /\ (m + 1) < j\ /\ (n + 1)$
$\qquad \Leftrightarrow i * (n + 1) < j * (m + 1)$

ℝ_minus_frac_less_frac_thm

$\vdash \forall\ i\ j\ m\ n \bullet\ \sim (i\ /\ (m + 1)) < j\ /\ (n + 1) \Leftrightarrow 0 < i + j$

ℝ_frac_less_minus_frac_thm

$\vdash \forall\ i\ j\ m\ n \bullet\ \neg\ i\ /\ (m + 1) < \sim (j\ /\ (n + 1))$

ℝ_0_≤_frac_thm

$\vdash \forall\ i\ m \bullet\ 0. \leq i\ /\ (m + 1)$

ℝ_abs_frac_thm

$\vdash \forall\ i\ m \bullet\ Abs\ (i\ /\ (m + 1)) = i\ /\ (m + 1)$

ℝ_abs_minus_thm

$\vdash \forall\ x \bullet\ Abs\ (\sim x) = Abs\ x$

$\mathbf{Max_R}$_consistent

$\vdash Consistent$
$\quad (\lambda\ Max'_R$
$\qquad \bullet\ (\forall\ x \bullet\ Max'_R\ [x] = x)$
$\qquad\quad \wedge (\forall\ x\ y\ L$

$\bullet\ Max'_R\ (Cons\ x\ (Cons\ y\ L))$
$= (if\ x < Max'_R\ (Cons\ y\ L)$
$then\ Max'_R\ (Cons\ y\ L)$
$else\ x)))$

$\mathbf{Min_R}$_consistent

$\vdash Consistent$
$(\lambda\ Min'_R$
$\bullet\ (\forall\ x \bullet\ Min'_R\ [x] = x)$
$\wedge\ (\forall\ x\ y\ L$
$\bullet\ Min'_R\ (Cons\ x\ (Cons\ y\ L))$
$= (if\ x > Min'_R\ (Cons\ y\ L)$
$then\ Min'_R\ (Cons\ y\ L)$
$else\ x)))$

$\mathbb{R}$_max_cons_thm

$\vdash \forall\ x\ L$
$\bullet\ Max_R\ (Cons\ x\ L)$
$= (if\ L = []$
$then\ x$
$else\ if\ x < Max_R\ L$
$then\ Max_R\ L$
$else\ x)$

$\mathbb{R}$_max_conv_thm

$\vdash \forall\ x\ y\ L$
$\bullet\ Max_R\ (Cons\ x\ (Cons\ y\ L))$
$= (if\ x < y$
$then\ Max_R\ (Cons\ y\ L)$
$else\ Max_R\ (Cons\ x\ L))$

$\mathbb{R}$_min_cons_thm

$\vdash \forall\ x\ L$
$\bullet\ Min_R\ (Cons\ x\ L)$
$= (if\ L = []$
$then\ x$
$else\ if\ Min_R\ L < x$
$then\ Min_R\ L$
$else\ x)$

$\mathbb{R}$_min_conv_thm

$\vdash \forall\ x\ y\ L$
$\bullet\ Min_R\ (Cons\ x\ (Cons\ y\ L))$
$= (if\ x < y$
$then\ Min_R\ (Cons\ x\ L)$
$else\ Min_R\ (Cons\ y\ L))$

$\mathbf{\hat{}_Z}$_consistent

$\vdash Consistent$
$(\lambda\ \hat{}'_Z$
$\bullet\ \forall\ x\ m$
$\bullet\ \hat{}'_Z\ x\ (\mathbb{N}\mathbb{Z}\ m) = x\ \hat{}\ m$
$\wedge\ \hat{}'_Z\ x\ (\sim (\mathbb{N}\mathbb{Z}\ (m + 1)))$
$= (x\ \hat{}\ (m + 1))^{-1})$

$\mathbb{Z}\mathbb{R}$_consistent

$\vdash Consistent$
$(\lambda\ \$"\mathbb{Z}\mathbb{R}'"$

$\bullet$ $"ZR'" (NZ 0) = $0.$
$\wedge$ $"ZR'" (NZ 1) = $1.$
$\wedge$ ($\forall$ i j
$\bullet$ $"ZR'" ($i$ + j) = $"ZR'" i + $"ZR'" j))

ℤℝ_plus_homomorphism_thm
⊢ ∀ i j• ℤℝ (i + j) = ℤℝ i + ℤℝ j

ℤℝ_minus_thm ⊢ ∀ i• ℤℝ (∼ i) = ∼ (ℤℝ i)

ℤℝ_ℕℤ_thm ⊢ ∀ m• ℤℝ (ℕℤ m) = ℕℝ m ∧ ℤℝ (∼ (ℕℤ m)) = ∼ (ℕℝ m)

ℤℝ_times_homomorphism_thm
⊢ ∀ i j• ℤℝ (i ∗ j) = ℤℝ i ∗ ℤℝ j

9.1.23 The Theory ℕ

9.1.23.1 Parents

pair

9.1.23.2 Children

list

9.1.23.3 Constants

Is_ℕ_Rep	$IND \to BOOL$
Suc	$\mathbb{N} \to \mathbb{N}$
Zero	$\mathbb{N}$
$\$+$	$\mathbb{N} \to \mathbb{N} \to \mathbb{N}$
$\$\leq$	$\mathbb{N} \to \mathbb{N} \to BOOL$
$\$\geq$	$\mathbb{N} \to \mathbb{N} \to BOOL$
$\$<$	$\mathbb{N} \to \mathbb{N} \to BOOL$
$\$>$	$\mathbb{N} \to \mathbb{N} \to BOOL$
$\$*$	$\mathbb{N} \to \mathbb{N} \to \mathbb{N}$
$**Mod**	$\mathbb{N} \to \mathbb{N} \to \mathbb{N}$
$**Div**	$\mathbb{N} \to \mathbb{N} \to \mathbb{N}$
$\$-$	$\mathbb{N} \to \mathbb{N} \to \mathbb{N}$

9.1.23.4 Types

ℕ

9.1.23.5 Fixity

Left Infix 305:

$-$

Left Infix 315:

Div **Mod**

Right Infix 210:

$<$ $>$ $\leq$ $\geq$

Right Infix 300:

$+$

Right Infix 310:

$*$

9.1.23.6 Definitions

Is_ℕ_Rep

is_ℕ_rep_def $\vdash \exists\ zero\ suc$
$\bullet\ (Is\_\mathbb{N}\_Rep\ zero$
$\quad \land (\forall\ n \bullet\ Is\_\mathbb{N}\_Rep\ n \Rightarrow Is\_\mathbb{N}\_Rep\ (suc\ n)))$
$\quad \land (\forall\ n \bullet\ Is\_\mathbb{N}\_Rep\ n \Rightarrow \neg\ suc\ n = zero)$
$\quad \land\ OneOne\ suc$
$\quad \land (\forall\ p$
$\quad \bullet\ p\ zero \land (\forall\ m \bullet\ p\ m \Rightarrow p\ (suc\ m))$
$\quad\quad \Rightarrow (\forall\ n \bullet\ Is\_\mathbb{N}\_Rep\ n \Rightarrow p\ n))$

ℕ

ℕ_def $\vdash \exists\ f \bullet\ TypeDefn\ Is\_\mathbb{N}\_Rep\ f$

Zero

Suc

zero_suc_def $\vdash (\forall\ n \bullet\ \neg\ Suc\ n = Zero)$
$\land\ OneOne\ Suc$
$\land (\forall\ p$
$\bullet\ p\ Zero \land (\forall\ m \bullet\ p\ m \Rightarrow p\ (Suc\ m)) \Rightarrow (\forall\ n \bullet\ p\ n))$

+

plus_def $\vdash \forall\ m\ n$
$\bullet\ 0 + n = n$
$\quad \land (m + 1) + n = (m + n) + 1$
$\quad \land\ Suc\ m = m + 1$

≤

≤_def $\vdash \forall\ m\ n \bullet\ m \le n \Leftrightarrow (\exists\ i \bullet\ m + i = n)$

≥

≥_def $\vdash \forall\ m\ n \bullet\ m \ge n \Leftrightarrow n \le m$

<

less_def $\vdash \forall\ m\ n \bullet\ m < n \Leftrightarrow m + 1 \le n$

>

greater_def $\vdash \forall\ m\ n \bullet\ m > n \Leftrightarrow n < m$

*****

times_def $\vdash \forall\ m\ n \bullet\ 0 * n = 0 \land (m + 1) * n = m * n + n$

Mod

mod_def $\vdash \forall\ m\ n$
$\bullet\ 0 < n$
$\quad \Rightarrow 0\ Mod\ n = 0$
$\quad\quad \land (m + 1)\ Mod\ n$
$\quad\quad\quad = (if\ m\ Mod\ n + 1 < n$
$\quad\quad\quad\quad then\ m\ Mod\ n + 1$
$\quad\quad\quad\quad else\ 0)$

Div

div_def $\vdash \forall\ m\ n$
$\bullet\ 0 < n$
$\quad \Rightarrow 0\ Div\ n = 0$
$\quad\quad \land (m + 1)\ Div\ n$
$\quad\quad\quad = (if\ m\ Mod\ n + 1 < n$
$\quad\quad\quad\quad then\ m\ Div\ n$
$\quad\quad\quad\quad else\ m\ Div\ n + 1)$

−

minus_def $\vdash \forall\ m\ n \bullet\ (m + n) - n = m$

9.1.23.7 Theorems

induction_thm
$\vdash \forall\ p \bullet\ p\ 0 \land (\forall\ m \bullet\ p\ m \Rightarrow p\ (m + 1)) \Rightarrow (\forall\ n \bullet\ p\ n)$

¬_plus1_thm $\vdash \forall\ n \bullet\ \neg\ n + 1 = 0$

one_one_plus1_thm
$\vdash \forall\ x1\ x2 \bullet\ x1 + 1 = x2 + 1 \Rightarrow x1 = x2$

prim_rec_thm $\vdash \forall\ z\ s \bullet\ \exists_1\ f \bullet\ f\ 0 = z \land (\forall\ n \bullet\ f\ (n + 1) = s\ (f\ n)\ n)$

plus_assoc_thm
$\vdash \forall\ i\ m\ n \bullet\ (i + m) + n = i + m + n$

plus_assoc_thm1
$\vdash \forall\ i\ m\ n \bullet\ i + m + n = (i + m) + n$

plus_comm_thm
$\vdash \forall\ m\ n \bullet\ m + n = n + m$

plus_order_thm
$\vdash \forall\ i\ m\ n$
$\bullet\ m + i = i + m$
$\land (i + m) + n = i + m + n$
$\land m + i + n = i + m + n$

plus_clauses $\vdash \forall\ m\ n\ i$
$\bullet\ (m + i = n + i \Leftrightarrow m = n)$
$\land (i + m = n + i \Leftrightarrow m = n)$
$\land (m + i = i + n \Leftrightarrow m = n)$
$\land (i + m = i + n \Leftrightarrow m = n)$
$\land (m + i = i \Leftrightarrow m = 0)$
$\land (i + m = i \Leftrightarrow m = 0)$
$\land (i = i + n \Leftrightarrow n = 0)$
$\land (i = n + i \Leftrightarrow n = 0)$
$\land (m + i = 0 \Leftrightarrow m = 0 \land i = 0)$
$\land (0 = m + i \Leftrightarrow m = 0 \land i = 0)$
$\land (m + 0 = m \land 0 + m = m)$
$\land \neg\ 1 = 0$
$\land \neg\ 0 = 1$

≤_trans_thm $\vdash \forall\ m\ i\ n \bullet\ m \le i \land i \le n \Rightarrow m \le n$

less_trans_thm
$\vdash \forall\ m\ i\ n \bullet\ m < i \land i < n \Rightarrow m < n$

≤_clauses $\vdash \forall\ m\ n\ i$
$\bullet\ (m + i \le n + i \Leftrightarrow m \le n)$
$\land (i + m \le n + i \Leftrightarrow m \le n)$
$\land (m + i \le i + n \Leftrightarrow m \le n)$
$\land (i + m \le i + n \Leftrightarrow m \le n)$
$\land (m + i \le i \Leftrightarrow m = 0)$
$\land (i + m \le i \Leftrightarrow m = 0)$
$\land (m + i \le 0 \Leftrightarrow m = 0 \land i = 0)$
$\land (m \le 0 \Leftrightarrow m = 0)$
$\land m \le m + i$
$\land m \le i + m$
$\land m \le m$
$\land 0 \le m$
$\land \neg\ 1 \le 0$

less_clauses $\vdash \forall\ m\ n\ i$
$\bullet\ (m + i < n + i \Leftrightarrow m < n)$

$\land (i + m < n + i \Leftrightarrow m < n)$
$\land (m + i < i + n \Leftrightarrow m < n)$
$\land (i + m < i + n \Leftrightarrow m < n)$
$\land (m < m + i \Leftrightarrow 0 < i)$
$\land (m < i + m \Leftrightarrow 0 < i)$
$\land \neg m + i < m$
$\land \neg m + i < i$
$\land \neg m < 0$
$\land \neg m < m$
$\land 0 < m + 1$
$\land 0 < 1 + m$
$\land 0 < 1$

ℕ_cases_thm $\vdash \forall m\bullet\ m = 0 \lor (\exists i\bullet\ m = i + 1)$

≤_cases_thm $\vdash \forall m\ n\bullet\ m \leq n \lor n \leq m$

≤_plus1_thm $\vdash \forall m\ n\bullet\ m \leq n + 1 \Leftrightarrow m = n + 1 \lor m \leq n$

plus1_≤_thm $\vdash \forall m\ n\bullet\ m + 1 \leq n \Leftrightarrow m \leq n \land \neg m = n$

¬_plus1_≤_thm
$\vdash \forall m\ n\bullet\ \neg m + 1 \leq n \Leftrightarrow n \leq m$

less_cases_thm
$\vdash \forall m\ n\bullet\ m < n \lor m = n \lor n < m$

¬_less_plus1_thm
$\vdash \forall m\ n\bullet\ \neg m < n + 1 \Leftrightarrow n < m$

less_plus1_thm
$\vdash \forall m\ n\bullet\ m < n + 1 \Leftrightarrow m = n \lor m < n$

plus1_less_thm
$\vdash \forall m\ n\bullet\ m + 1 < n \Leftrightarrow m < n \land \neg m + 1 = n$

≤_antisym_thm
$\vdash \forall m\ n\bullet\ m \leq n \land n \leq m \Leftrightarrow m = n$

less_irrefl_thm
$\vdash \forall m\ n\bullet\ \neg (m < n \land n < m)$

cov_induction_thm
$\vdash \forall p\bullet\ (\forall n\bullet\ (\forall m\bullet\ m < n \Rightarrow p\ m) \Rightarrow p\ n) \Rightarrow (\forall n\bullet\ p\ n)$

less_well_order_thm
$\vdash \forall p\bullet\ (\exists i\bullet\ p\ i) \Leftrightarrow (\exists m\bullet\ p\ m \land (\forall i\bullet\ p\ i \Rightarrow \neg i < m))$

¬_less_thm $\vdash \forall m\ n\bullet\ \neg m < n \Leftrightarrow n \leq m$

¬_≤_thm $\vdash \forall m\ n\bullet\ \neg m \leq n \Leftrightarrow n < m$

≤_well_order_thm
$\vdash \forall p\bullet\ (\exists i\bullet\ p\ i) \Leftrightarrow (\exists m\bullet\ p\ m \land (\forall i\bullet\ p\ i \Rightarrow m \leq i))$

≤_least_upper_bound_thm
$\vdash \forall p$
$\bullet\ (\exists i\bullet\ p\ i) \land (\exists n\bullet\ \forall j\bullet\ p\ j \Rightarrow j \leq n)$
$\Leftrightarrow (\exists m\bullet\ p\ m \land (\forall j\bullet\ p\ j \Rightarrow j \leq m))$

minimum_¬_thm
$\vdash \forall p\ b$
$\bullet\ p\ 0 \land \neg p\ b$
$\Rightarrow (\exists m\bullet\ (\forall n\bullet\ n \leq m \Rightarrow p\ n) \land \neg p\ (m + 1))$

times_comm_thm
$\vdash \forall m\ n\bullet\ m * n = n * m$

times_assoc_thm
$\vdash \forall i\ m\ n\bullet\ (i * m) * n = i * m * n$

times_plus_distrib_thm

$\vdash \forall\ i\ m\ n$
$\quad\bullet\ (i + m) * n = i * n + m * n$
$\qquad\wedge\ i * (m + n) = i * m + i * n$

times_clauses
$\vdash \forall\ m\bullet\ m * \mathit{0} = \mathit{0} \wedge \mathit{0} * m = \mathit{0} \wedge m * \mathit{1} = m \wedge \mathit{1} * m = m$

mod_less_thm $\vdash \forall\ m\ n\bullet\ \mathit{0} < n \Rightarrow m\ \mathit{Mod}\ n < n$

div_mod_thm $\vdash \forall\ m\ n\bullet\ \mathit{0} < n \Rightarrow m = m\ \mathit{Div}\ n * n + m\ \mathit{Mod}\ n$

div_mod_unique_thm
$\vdash \forall\ m\ n\ d\ r$
$\quad\bullet\ r < n \Rightarrow m = d * n + r \Rightarrow d = m\ \mathit{Div}\ n \wedge r = m\ \mathit{Mod}\ n$

minus_clauses
$\vdash \forall\ m\ n$
$\quad\bullet\ m - m = \mathit{0}$
$\qquad\wedge\ m - \mathit{0} = m$
$\qquad\wedge\ (m + n) - n = m$
$\qquad\wedge\ (m + n) - m = n$

9.1.24 The Theory $\mathbb{Z}$

9.1.24.1 Parents

sets

9.1.24.2 Children

dyadic

9.1.24.3 Constants

Is_$\mathbb{Z}$_Rep	$(\mathbb{N} \times \mathbb{N})\ SET \to BOOL$
$\mathbb{N}\mathbb{Z}$	$\mathbb{N} \to \mathbb{Z}$
$\$\sim_{\mathbf{Z}}$	$\mathbb{Z} \to \mathbb{Z}$
$\$+_{\mathbf{Z}}$	$\mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$
$\$-_{\mathbf{Z}}$	$\mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$
$\$*_{\mathbf{Z}}$	$\mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$
$\$\leq_{\mathbf{Z}}$	$\mathbb{Z} \to \mathbb{Z} \to BOOL$
$\$<_{\mathbf{Z}}$	$\mathbb{Z} \to \mathbb{Z} \to BOOL$
$\$\geq_{\mathbf{Z}}$	$\mathbb{Z} \to \mathbb{Z} \to BOOL$
$\$>_{\mathbf{Z}}$	$\mathbb{Z} \to \mathbb{Z} \to BOOL$
$\$\mathbf{Abs}_{\mathbf{Z}}$	$\mathbb{Z} \to \mathbb{Z}$
$\$\mathbf{Mod}_{\mathbf{Z}}$	$\mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$
$\$\mathbf{Div}_{\mathbf{Z}}$	$\mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$

9.1.24.4 Aliases

$+$	$\$+_Z : \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$
$-$	$\$-_Z : \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$
$\sim$	$\$\sim_Z : \mathbb{Z} \to \mathbb{Z}$
$*$	$\$*_Z : \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$
$\leq$	$\$\leq_Z : \mathbb{Z} \to \mathbb{Z} \to BOOL$
$<$	$\$<_Z : \mathbb{Z} \to \mathbb{Z} \to BOOL$
$\geq$	$\$\geq_Z : \mathbb{Z} \to \mathbb{Z} \to BOOL$
$>$	$\$>_Z : \mathbb{Z} \to \mathbb{Z} \to BOOL$
Abs	$\$Abs_Z : \mathbb{Z} \to \mathbb{Z}$
Div	$\$Div_Z : \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$
Mod	$\$Mod_Z : \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$

9.1.24.5 Types

$\mathbb{Z}$

9.1.24.6 Fixity

Left Infix 305:
$-_{\mathbf{Z}}$
Left Infix 315:
$\mathbf{Div_Z}$ $\mathbf{Mod_Z}$
Right Infix 210:
$<_{\mathbf{Z}}$ $>_{\mathbf{Z}}$ $\leq_{\mathbf{Z}}$ $\geq_{\mathbf{Z}}$
Right Infix 300:
$+_{\mathbf{Z}}$
Right Infix 310:
$*_{\mathbf{Z}}$
Prefix 350: $\mathbf{Abs_Z}$ $\sim_{\mathbf{Z}}$

9.1.24.7 Definitions

Is_ℤ_Rep $\vdash \forall\ a\bullet\ Is\_\mathbb{Z}\_Rep\ a \Leftrightarrow (\exists\ m\ n\bullet\ a = \{(x,\ y)|m + y = n + x\})$
ℤ
ℤ_def $\vdash \exists\ f\bullet\ TypeDefn\ Is\_\mathbb{Z}\_Rep\ f$
$+_{\mathbf{Z}}$
$\sim_{\mathbf{Z}}$
ℕℤ $\vdash$ *ConstSpec*

$$
\begin{array}{l}
(\lambda\ (+'_Z,\ \sim'_Z,\ \$\text{"}\mathbb{N}\mathbb{Z}'\text{"})\\
\quad \bullet\ (\forall\ i\ j\ k\\
\qquad \bullet\ +'_Z\ (+'_Z\ i\ j)\ k = +'_Z\ i\ (+'_Z\ j\ k)\\
\qquad\quad \wedge\ +'_Z\ i\ j = +'_Z\ j\ i\\
\qquad\quad \wedge\ +'_Z\ i\ (\sim'_Z\ i) = \$\text{"}\mathbb{N}\mathbb{Z}'\text{"}\ 0\\
\qquad\quad \wedge\ +'_Z\ i\ (\$\text{"}\mathbb{N}\mathbb{Z}'\text{"}\ 0) = i)\\
\qquad \wedge\ (\forall\ m\ n\\
\qquad \bullet\ +'_Z\ (\$\text{"}\mathbb{N}\mathbb{Z}'\text{"}\ m)\ (\$\text{"}\mathbb{N}\mathbb{Z}'\text{"}\ n)\\
\qquad\quad = \$\text{"}\mathbb{N}\mathbb{Z}'\text{"}\ (m + n))\\
\qquad \wedge\ OneOne\ \$\text{"}\mathbb{N}\mathbb{Z}'\text{"}\\
\qquad \wedge\ (\forall\ i\\
\qquad \bullet\ \exists\ m\bullet\ i = \$\text{"}\mathbb{N}\mathbb{Z}'\text{"}\ m \vee i = \sim'_Z\ (\$\text{"}\mathbb{N}\mathbb{Z}'\text{"}\ m)))\\
(\$+,\ \sim,\ \mathbb{N}\mathbb{Z})
\end{array}
$$

$-_{\mathbf{Z}}$ $\vdash \forall\ i\ j\bullet\ i - j = i + \sim j$
$*_{\mathbf{Z}}$ $\vdash$ *ConstSpec*

$$
\begin{array}{l}
(\lambda\ *'_Z\\
\quad \bullet\ \forall\ i\ j\ k\\
\qquad \bullet\ *'_Z\ i\ (j + k) = *'_Z\ i\ j + *'_Z\ i\ k\\
\qquad\quad \wedge\ *'_Z\ i\ (\mathbb{N}\mathbb{Z}\ 1) = i)\\
\$*
\end{array}
$$

$\leq_{\mathbf{Z}}$ $\vdash \forall\ i\ j\bullet\ i \leq j \Leftrightarrow (\exists\ m\bullet\ i + \mathbb{N}\mathbb{Z}\ m = j)$
$<_{\mathbf{Z}}$ $\vdash \forall\ i\ j\bullet\ i < j \Leftrightarrow i + \mathbb{N}\mathbb{Z}\ 1 \leq j$
$\geq_{\mathbf{Z}}$ $\vdash \forall\ i\ j\bullet\ i \geq j \Leftrightarrow j \leq i$
$>_{\mathbf{Z}}$ $\vdash \forall\ i\ j\bullet\ i > j \Leftrightarrow j < i$
$\mathbf{Abs_Z}$ $\vdash \forall\ i\bullet\ Abs\ i = (if\ \mathbb{N}\mathbb{Z}\ 0 \leq i\ then\ i\ else \sim i)$
$\mathbf{Div_Z}$
$\mathbf{Mod_Z}$ $\vdash$ *ConstSpec*

$$
\begin{array}{l}
(\lambda\ (Div'_Z,\ Mod'_Z)\\
\quad \bullet\ \forall\ i\ j
\end{array}
$$

$$\begin{array}{l}
\bullet\ \neg\ j = \mathbb{NZ}\ 0 \\
\quad \Rightarrow i = Div'_Z\ i\ j * j + Mod'_Z\ i\ j \\
\quad\ \ \wedge\ \mathbb{NZ}\ 0 \leq Mod'_Z\ i\ j \\
\quad\ \ \wedge\ Mod'_Z\ i\ j < Abs\ j)
\end{array}$$

(\$Div, \$Mod)

9.1.24.8 Theorems

is_ℤ_rep_consistent_thm

$\vdash \exists\ a\bullet\ Is\_\mathbb{Z}\_Rep\ a$

$+_Z$_consistent
$\sim_Z$_consistent
ℕℤ_consistent

$$\begin{array}{l}
\vdash Consistent \\
\quad (\lambda\ (+'_Z,\ \sim'_Z,\ \$\text{"}\mathbb{NZ}'\text{"}) \\
\quad\ \bullet\ (\forall\ i\ j\ k \\
\quad\quad \bullet\ +'_Z\ (+'_Z\ i\ j)\ k = +'_Z\ i\ (+'_Z\ j\ k) \\
\quad\quad\quad \wedge\ +'_Z\ i\ j = +'_Z\ j\ i \\
\quad\quad\quad \wedge\ +'_Z\ i\ (\sim'_Z\ i) = \$\text{"}\mathbb{NZ}'\text{"}\ 0 \\
\quad\quad\quad \wedge\ +'_Z\ i\ (\$\text{"}\mathbb{NZ}'\text{"}\ 0) = i) \\
\quad\quad \wedge\ (\forall\ m\ n \\
\quad\quad \bullet\ +'_Z\ (\$\text{"}\mathbb{NZ}'\text{"}\ m)\ (\$\text{"}\mathbb{NZ}'\text{"}\ n) \\
\quad\quad\quad = \$\text{"}\mathbb{NZ}'\text{"}\ (m + n)) \\
\quad\quad \wedge\ OneOne\ \$\text{"}\mathbb{NZ}'\text{"} \\
\quad\quad \wedge\ (\forall\ i \\
\quad\quad \bullet\ \exists\ m\bullet\ i = \$\text{"}\mathbb{NZ}'\text{"}\ m \vee i = \sim'_Z\ (\$\text{"}\mathbb{NZ}'\text{"}\ m)))
\end{array}$$

ℤ_plus_comm_thm

$\vdash \forall\ i\ j\bullet\ i + j = j + i$

ℤ_plus_assoc_thm

$\vdash \forall\ i\ j\ k\bullet\ (i + j) + k = i + j + k$

ℤ_plus_assoc_thm1

$\vdash \forall\ i\ j\ k\bullet\ i + j + k = (i + j) + k$

ℤ_plus_order_thm

$$\begin{array}{l}
\vdash \forall\ i\ j\ k \\
\quad \bullet\ j + i = i + j \\
\quad\quad \wedge\ (i + j) + k = i + j + k \\
\quad\quad \wedge\ j + i + k = i + j + k
\end{array}$$

ℤ_cases_thm $\vdash \forall\ i\bullet\ \exists\ m\bullet\ i = \mathbb{NZ}\ m \vee i = \sim\ (\mathbb{NZ}\ m)$

ℤ_plus0_thm $\vdash \forall\ i\bullet\ i + \mathbb{NZ}\ 0 = i \wedge \mathbb{NZ}\ 0 + i = i$

ℤ_plus_minus_thm

$\vdash \forall\ i\bullet\ i + \sim\ i = \mathbb{NZ}\ 0 \wedge \sim\ i + i = \mathbb{NZ}\ 0$

ℤ_eq_thm $\vdash \forall\ i\ j\bullet\ i = j \Leftrightarrow i + \sim\ j = \mathbb{NZ}\ 0$

ℕℤ_plus_homomorphism_thm

$\vdash \forall\ m\ n\bullet\ \mathbb{NZ}\ (m + n) = \mathbb{NZ}\ m + \mathbb{NZ}\ n$

ℤ_minus_clauses

$$\begin{array}{l}
\vdash \forall\ i\ j \\
\quad \bullet\ \sim\ (\sim\ i) = i \\
\quad\quad \wedge\ i + \sim\ i = \mathbb{NZ}\ 0 \\
\quad\quad \wedge\ \sim\ i + i = \mathbb{NZ}\ 0 \\
\quad\quad \wedge\ \sim\ (i + j) = \sim\ i + \sim\ j \\
\quad\quad \wedge\ \sim\ (\mathbb{NZ}\ 0) = \mathbb{NZ}\ 0
\end{array}$$

ℤ_cases_thm1 $\vdash \forall\ i\bullet\ \exists\ m\bullet\ i = ℕℤ\ m \lor i = \sim (ℕℤ\ (m + 1))$

ℤ_induction_thm

$\vdash \forall\ p$
$\bullet\ p\ (ℕℤ\ 1)$
$\quad \land (\forall\ i\bullet\ p\ i \Rightarrow p\ (\sim i))$
$\quad \land (\forall\ i\ j\bullet\ p\ i \land p\ j \Rightarrow p\ (i + j))$
$\quad \Rightarrow (\forall\ i\bullet\ p\ i)$

ℕℤ_one_one_thm

$\vdash \forall\ m\ n\bullet\ ℕℤ\ m = ℕℤ\ n \Leftrightarrow m = n$

ℤ_plus_clauses

$\vdash \forall\ i\ j\ k$
$\bullet\ (i + k = j + k \Leftrightarrow i = j)$
$\quad \land (k + i = j + k \Leftrightarrow i = j)$
$\quad \land (i + k = k + j \Leftrightarrow i = j)$
$\quad \land (k + i = k + j \Leftrightarrow i = j)$
$\quad \land (i + k = k \Leftrightarrow i = ℕℤ\ 0)$
$\quad \land (k + i = k \Leftrightarrow i = ℕℤ\ 0)$
$\quad \land (k = k + j \Leftrightarrow j = ℕℤ\ 0)$
$\quad \land (k = j + k \Leftrightarrow j = ℕℤ\ 0)$
$\quad \land i + ℕℤ\ 0 = i$
$\quad \land ℕℤ\ 0 + i = i$
$\quad \land \lnot ℕℤ\ 1 = ℕℤ\ 0$
$\quad \land \lnot ℕℤ\ 0 = ℕℤ\ 1$

ℤ_≤_≤_0_thm $\vdash \forall\ i\ j\bullet\ i \le j \Leftrightarrow i + \sim j \le ℕℤ\ 0$

ℤ_minus_≤_thm

$\vdash \forall\ i\ j\bullet\ \sim i \le \sim j \Leftrightarrow j \le i$

ℤ_≤_minus_thm

$\vdash \forall\ i\ j\bullet\ i \le j \Leftrightarrow \sim j \le \sim i$

ℤ_≤_clauses $\vdash \forall\ i\ j\ k$
$\bullet\ (i + k \le j + k \Leftrightarrow i \le j)$
$\quad \land (k + i \le j + k \Leftrightarrow i \le j)$
$\quad \land (i + k \le k + j \Leftrightarrow i \le j)$
$\quad \land (k + i \le k + j \Leftrightarrow i \le j)$
$\quad \land (i + k \le k \Leftrightarrow i \le ℕℤ\ 0)$
$\quad \land (k + i \le k \Leftrightarrow i \le ℕℤ\ 0)$
$\quad \land (k \le k + j \Leftrightarrow ℕℤ\ 0 \le j)$
$\quad \land (k \le j + k \Leftrightarrow ℕℤ\ 0 \le j)$
$\quad \land i \le i$
$\quad \land \lnot ℕℤ\ 1 \le ℕℤ\ 0$
$\quad \land ℕℤ\ 0 \le ℕℤ\ 1$

ℕℤ_≤_thm $\vdash \forall\ m\ n\bullet\ ℕℤ\ m \le ℕℤ\ n \Leftrightarrow m \le n$

$*_Z$_consistent

$\vdash$ *Consistent*
$\quad (\lambda\ *'_Z$
$\quad \bullet\ \forall\ i\ j\ k$
$\quad\quad \bullet\ *'_Z\ i\ (j + k) = *'_Z\ i\ j + *'_Z\ i\ k$
$\quad\quad\quad \land *'_Z\ i\ (ℕℤ\ 1) = i)$

ℕℤ_times_homomorphism_thm

$\vdash \forall\ m\ n\bullet\ ℕℤ\ (m * n) = ℕℤ\ m * ℕℤ\ n$

ℤ_times_minus_thm

$\vdash \forall\ i\ j$

$\bullet \sim i * j = \sim (i * j)$
$\quad \wedge\ i * \sim j = \sim (i * j)$
$\quad \wedge \sim i * \sim j = i * j$

ℤ_times_comm_thm
$\vdash \forall\ i\ j \bullet\ i * j = j * i$

ℤ_times_assoc_thm
$\vdash \forall\ i\ j\ k \bullet\ (i * j) * k = i * j * k$

Div$_\mathbf{Z}$_consistent
Mod$_\mathbf{Z}$_consistent
$\vdash Consistent$
$\quad (\lambda\ (Div'_Z,\ Mod'_Z)$
$\quad\quad \bullet \forall\ i\ j$
$\quad\quad\quad \bullet \neg\ j = ℕℤ\ 0$
$\quad\quad\quad\quad \Rightarrow i = Div'_Z\ i\ j * j + Mod'_Z\ i\ j$
$\quad\quad\quad\quad\quad \wedge\ ℕℤ\ 0 \le Mod'_Z\ i\ j$
$\quad\quad\quad\quad\quad \wedge\ Mod'_Z\ i\ j < Abs\ j)$

ℕℤ_plus_homomorphism_thm1
$\vdash \forall\ m\ n \bullet\ ℕℤ\ m + ℕℤ\ n = ℕℤ\ (m + n)$

ℤ_ℕ_induction_thm
$\vdash \forall\ p$
$\quad \bullet\ p\ (ℕℤ\ 0) \wedge (\forall\ i \bullet\ ℕℤ\ 0 \le i \wedge p\ i \Rightarrow p\ (i + ℕℤ\ 1))$
$\quad\quad \Rightarrow (\forall\ m \bullet\ ℕℤ\ 0 \le m \Rightarrow p\ m)$

ℤ_ℕ_plus_thm $\vdash \forall\ i\ j \bullet\ ℕℤ\ 0 \le i \wedge ℕℤ\ 0 \le j \Rightarrow ℕℤ\ 0 \le i + j$

ℤ_ℕ_plus1_thm
$\vdash \forall\ i \bullet\ ℕℤ\ 0 \le i \Rightarrow ℕℤ\ 0 \le i + ℕℤ\ 1$

ℤ_minus_thm $\vdash \forall\ i\ j$
$\quad \bullet \sim (\sim i) = i$
$\quad\quad \wedge\ i + \sim i = ℕℤ\ 0$
$\quad\quad \wedge \sim i + i = ℕℤ\ 0$
$\quad\quad \wedge \sim (i + j) = \sim i + \sim j$
$\quad\quad \wedge \sim (ℕℤ\ 0) = ℕℤ\ 0$

ℤ_ℕ_cases_thm
$\vdash \forall\ i$
$\quad \bullet\ ℕℤ\ 0 \le i$
$\quad\quad \Rightarrow i = ℕℤ\ 0 \vee (\exists\ j \bullet\ ℕℤ\ 0 \le j \wedge i = j + ℕℤ\ 1)$

ℤ_ℕ_¬_minus_thm
$\vdash \forall\ i \bullet\ ℕℤ\ 0 \le i \Rightarrow i = ℕℤ\ 0 \vee \neg\ ℕℤ\ 0 \le \sim i$

ℤ_¬_ℕ_thm $\vdash \forall\ i \bullet \neg\ ℕℤ\ 0 \le i \Rightarrow ℕℤ\ 0 \le \sim i$

ℤ_plus_eq_thm
$\vdash \forall\ i\ j\ k \bullet\ i + j = k \Leftrightarrow i = k + \sim j$

ℤ_ℕ_¬_plus1_thm
$\vdash \forall\ i \bullet\ ℕℤ\ 0 \le i \Rightarrow \neg\ i + ℕℤ\ 1 = ℕℤ\ 0$

ℤ_times_assoc_thm1
$\vdash \forall\ i\ j\ k \bullet\ i * j * k = (i * j) * k$

ℤ_times_order_thm
$\vdash \forall\ i\ j\ k$
$\quad \bullet\ j * i = i * j$
$\quad\quad \wedge\ (i * j) * k = i * j * k$
$\quad\quad \wedge\ j * i * k = i * j * k$

ℕℤ_times_homomorphism_thm1
$\vdash \forall\ m\ n \bullet\ ℕℤ\ m * ℕℤ\ n = ℕℤ\ (m * n)$

ℤ_times1_thm $\vdash \forall\ i \bullet\ i * ℕℤ\ 1 = i \land ℕℤ\ 1 * i = i$

ℤ_times_plus_distrib_thm
$\vdash \forall\ i\ j\ k$
$\bullet\ i * (j + k) = i * j + i * k$
$\land\ (i + j) * k = i * k + j * k$

ℤ_times0_thm $\vdash \forall\ i \bullet\ ℕℤ\ 0 * i = ℕℤ\ 0 \land i * ℕℤ\ 0 = ℕℤ\ 0$

ℤ_eq_thm1 $\vdash \forall\ i\ j \bullet\ i = j \Leftrightarrow \sim i + j = ℕℤ\ 0$

ℤ_times_eq_0_thm
$\vdash \forall\ i\ j \bullet\ i * j = ℕℤ\ 0 \Leftrightarrow i = ℕℤ\ 0 \lor j = ℕℤ\ 0$

ℤ_times_clauses
$\vdash \forall\ i\ j$
$\bullet\ ℕℤ\ 0 * i = ℕℤ\ 0$
$\land\ i * ℕℤ\ 0 = ℕℤ\ 0$
$\land\ i * ℕℤ\ 1 = i$
$\land\ ℕℤ\ 1 * i = i$

ℤ_ℕ_times_thm
$\vdash \forall\ i\ j \bullet\ ℕℤ\ 0 \le i \land ℕℤ\ 0 \le j \Rightarrow ℕℤ\ 0 \le i * j$

ℤ_≤_trans_thm
$\vdash \forall\ i\ j\ k \bullet\ i \le j \land j \le k \Rightarrow i \le k$

ℤ_≤_cases_thm
$\vdash \forall\ i\ j \bullet\ i \le j \lor j \le i$

ℤ_≤_refl_thm $\vdash \forall\ i \bullet\ i \le i$

ℤ_≤_≤_0_thm1 $\vdash \forall\ i\ j \bullet\ i \le j \Leftrightarrow ℕℤ\ 0 \le j + \sim i$

ℤ_≤_antisym_thm
$\vdash \forall\ i\ j \bullet\ i \le j \land j \le i \Rightarrow i = j$

ℤ_less_trans_thm
$\vdash \forall\ i\ j\ k \bullet\ i < j \land j < k \Rightarrow i < k$

ℤ_less_irrefl_thm
$\vdash \forall\ i\ j \bullet\ \neg\ (i < j \land j < i)$

ℤ_less_cases_thm
$\vdash \forall\ i\ j \bullet\ i < j \lor i = j \lor j < i$

ℕℤ_less_thm $\vdash \forall\ m\ n \bullet\ ℕℤ\ m < ℕℤ\ n \Leftrightarrow m < n$

ℤ_less_less_0_thm
$\vdash \forall\ i\ j \bullet\ i < j \Leftrightarrow i + \sim j < ℕℤ\ 0$

ℤ_less_less_0_thm1
$\vdash \forall\ i\ j \bullet\ i < j \Leftrightarrow ℕℤ\ 0 < j + \sim i$

ℤ_minus_less_thm
$\vdash \forall\ i\ j \bullet\ \sim i < \sim j \Leftrightarrow j < i$

ℤ_¬_less_thm $\vdash \forall\ i\ j \bullet\ \neg\ i < j \Leftrightarrow j \le i$

ℤ_¬_≤_thm $\vdash \forall\ i\ j \bullet\ \neg\ i \le j \Leftrightarrow j < i$

ℤ_≤_less_eq_thm
$\vdash \forall\ i\ j \bullet\ i \le j \Leftrightarrow i < j \lor i = j$

ℤ_less_≤_trans_thm
$\vdash \forall\ i\ j\ k \bullet\ i < j \land j \le k \Rightarrow i < k$

ℤ_≤_less_trans_thm
$\vdash \forall\ i\ j\ k \bullet\ i \le j \land j < k \Rightarrow i < k$

ℤ_minus_ℕ_≤_thm
$\vdash \forall\ i\ m \bullet\ i + \sim (ℕℤ\ m) \le i$

ℤ_≤_plus_ℕ_thm
$\vdash \forall\ i\ m \bullet\ i \le i + ℕℤ\ m$

ℤ_∈_ℕ_thm $\vdash \forall\ i \bullet\ ℕℤ\ 0 \le i \Leftrightarrow (\exists\ m \bullet\ i = ℕℤ\ m)$

ℤ_less_clauses

$\vdash \forall\ i\ j\ k$
$\quad \bullet\ (i + k < j + k \Leftrightarrow i < j)$
$\qquad \wedge\ (k + i < j + k \Leftrightarrow i < j)$
$\qquad \wedge\ (i + k < k + j \Leftrightarrow i < j)$
$\qquad \wedge\ (k + i < k + j \Leftrightarrow i < j)$
$\qquad \wedge\ (i + k < k \Leftrightarrow i < \mathbb{NZ}\ 0)$
$\qquad \wedge\ (k + i < k \Leftrightarrow i < \mathbb{NZ}\ 0)$
$\qquad \wedge\ (i < k + i \Leftrightarrow \mathbb{NZ}\ 0 < k)$
$\qquad \wedge\ (i < i + k \Leftrightarrow \mathbb{NZ}\ 0 < k)$
$\qquad \wedge\ \neg\ i < i$
$\qquad \wedge\ \mathbb{NZ}\ 0 < \mathbb{NZ}\ 1$
$\qquad \wedge\ \neg\ \mathbb{NZ}\ 1 < \mathbb{NZ}\ 0$

ℤ_ℕ_abs_thm $\vdash \forall\ m\bullet\ Abs\ (\mathbb{NZ}\ m) = \mathbb{NZ}\ m \wedge Abs\ ({\sim}\ (\mathbb{NZ}\ m)) = \mathbb{NZ}\ m$

ℤ_abs_thm $\vdash \forall\ i\bullet\ \mathbb{NZ}\ 0 \le i \Rightarrow Abs\ i = i \wedge Abs\ ({\sim}\ i) = i$

ℤ_abs_ℕ_thm $\vdash \forall\ i\bullet\ \mathbb{NZ}\ 0 \le Abs\ i$

ℤ_abs_eq_0_thm

$\vdash \forall\ i\bullet\ Abs\ i = \mathbb{NZ}\ 0 \Leftrightarrow i = \mathbb{NZ}\ 0$

ℤ_abs_minus_thm

$\vdash \forall\ i\bullet\ Abs\ ({\sim}\ i) = Abs\ i$

ℤ_ℕ_abs_minus_thm

$\vdash \forall\ i\ j$
$\quad \bullet\ \mathbb{NZ}\ 0 \le i \wedge \mathbb{NZ}\ 0 \le j \wedge j \le i \Rightarrow Abs\ (i + {\sim}\ j) \le i$

ℤ_abs_times_thm

$\vdash \forall\ i\ j\bullet\ Abs\ (i * j) = Abs\ i * Abs\ j$

ℤ_abs_plus_thm

$\vdash \forall\ i\ j\bullet\ Abs\ (i + j) \le Abs\ i + Abs\ j$

ℤ_div_mod_unique_lemma1

$\vdash \forall\ i\ j\bullet\ \mathbb{NZ}\ 0 \le i \wedge \mathbb{NZ}\ 0 \le j \wedge i * j < j \Rightarrow i = \mathbb{NZ}\ 0$

ℤ_div_mod_unique_lemma2

$\vdash \forall\ j\ d\ r$
$\quad \bullet\ \neg\ j = \mathbb{NZ}\ 0$
$\qquad \Rightarrow d * j + r = \mathbb{NZ}\ 0 \wedge \mathbb{NZ}\ 0 \le r \wedge r < Abs\ j$
$\qquad \Rightarrow d = \mathbb{NZ}\ 0 \wedge r = \mathbb{NZ}\ 0$

ℤ_div_mod_unique_lemma3

$\vdash \forall\ i\ j\ d\ r\ D\ R$
$\quad \bullet\ \neg\ j = \mathbb{NZ}\ 0$
$\qquad \Rightarrow D * j + R = d * j + r$
$\qquad\quad \wedge\ \mathbb{NZ}\ 0 \le r$
$\qquad\quad \wedge\ r \le R$
$\qquad\quad \wedge\ R < Abs\ j$
$\qquad \Rightarrow D = d \wedge R = r$

ℤ_div_mod_unique_thm

$\vdash \forall\ i\ j\ d\ r$
$\quad \bullet\ \neg\ j = \mathbb{NZ}\ 0$
$\qquad \Rightarrow (i = d * j + r \wedge \mathbb{NZ}\ 0 \le r \wedge r < Abs\ j$
$\qquad\quad \Leftrightarrow d = i\ Div\ j \wedge r = i\ Mod\ j)$

ℤ_≤_induction_thm

$\vdash \forall\ j\ p$
$\quad \bullet\ p\ j \wedge (\forall\ i\bullet\ j \le i \wedge p\ i \Rightarrow p\ (i + \mathbb{NZ}\ 1))$
$\qquad \Rightarrow (\forall\ i\bullet\ j \le i \Rightarrow p\ i)$

ℤ_cov_induction_thm

$\vdash \forall\ j\ p$
$\bullet\ (\forall\ i \bullet\ j \le i \land (\forall\ k \bullet\ j \le k \land k < i \Rightarrow p\ k) \Rightarrow p\ i)$
$\Rightarrow (\forall\ i \bullet\ j \le i \Rightarrow p\ i)$

ℤ_fun_∃_thm $\vdash \forall\ f\ g\ z$
$\bullet\ (\forall\ x \bullet\ g\ (f\ x) = x) \land (\forall\ y \bullet\ f\ (g\ y) = y)$
$\Rightarrow (\exists_1\ h$
$\bullet\ h\ (\mathbb{NZ}\ 0) = z$
$\land (\forall\ i \bullet\ h\ (i + \mathbb{NZ}\ 1) = f\ (h\ i))$
$\land (\forall\ i \bullet\ h\ (i - \mathbb{NZ}\ 1) = g\ (h\ i)))$

9.2 Theory Related ML Values

This section contains various theory related ML values (e.g. the value of theorems bound to ML names, or special tactics of proof contexts associated with the theory). Where a theorem or definition is bound to an ML name the value of the theorem is to be found in the theory listing, only the ML name is given below.

9.2.1 Basic Definitions And Axioms

SML
```
signature InitTheory = sig
        val bool_cases_axiom : THM;
        val η_axiom : THM;
        val infinity_axiom : THM;
        val ⇒_antisym_axiom : THM;
        val ε_axiom : THM;
end;
```

Description The signature *InitTheory* contains the definitions for the theory *init*(q.v). This contains the five primitive axioms of HOL.

SML
```
signature LogTheory = sig
        val f_def : THM;
        val one_one_def : THM;
        val onto_def : THM;
        val t_def : THM;
        val type_defn_def : THM;
        val ∃_def : THM;
        val ¬_def : THM;
        val ∧_def : THM;
        val ∀_def : THM;
        val ∨_def : THM;
end;
```

Description The signature *LogTheory* contains the definitions for the theory *log*(q.v.). This defines the basic predicate calculus connectives and other derived notions which are needed to state the axioms for HOL.

SML
```
signature MinTheory = sig end;
```

Description The signature *MinTheory* contains the definitions for the theory *min*. This introduces the primitive types and constants of HOL.

SML
```
signature MiscTheory = sig
        val cond_def : THM;
        val let_def : THM;
        val ∃₁_def : THM;
end;
```

Description The signature *MiscTheory* contains the initial definitions for the theory *misc*. This contains miscellaneous definitions and theorems required in constructing the proof development system. Further items may be saved in theory "misc", and bound to ML variables in other signatures and structures.

9.2.2 Pairs

SML
```
signature PairTheory = sig
```

Description This is the signature in which we declare theory "pair", and add a few theorems to theory "misc".

SML
```
val pair_clauses : THM;
end;
```

Description This single theorem collects all the externally useful results of the theory of pairs into one portmanteau theorem. It is saved in theory "pair" with its name as its key.

SML
```
val pair_ops_def : THM;
val curry_def : THM;
val uncurry_def : THM;
val is_pair_rep_def : THM;
```

Description These definitions are saved under their names as keys in theory "misc", and are the definitions for *Fst*, *Snd*, ,, and *IsPairRep*.

SML
```
val type_lemmas_thm : THM;
val one_one_thm : THM;
val ext_thm : THM;
val fun_rel_thm : THM;
```

Description These four theorems are saved under their names as keys in theory "misc". Their design is given in theory "misc", and not here, though they are implemented within the structure *PairTheory*.

9.2.3 Elementary Arithmetic

SML
```
signature ℕ = sig
```

Description This is the signature in which the theory "ℕ" is declared. This theory contains facts about the natural numbers (i.e.: *0*, *1*, *2*, ...) and their usual arithmetic operators.

SML
```
val cov_induction_tac : TERM -> TACTIC;
```

Description This tactic implements course-of-values induction over the natural numbers. To prove $t[x]$ it suffices to prove $t[x]$ on the assumption $\forall m \bullet m < x \Rightarrow t[m]$. (Course-of-values induction is also sometimes called as complete induction.) The term argument must be a variable of type ⌜:ℕ⌝ and must appear free in the conclusion of the goal but not in its assumptions.

Tactic

$$\frac{\{\ \Gamma\ \}\ t[x]}{strip\{\ulcorner\forall m\bullet\ m < x \Rightarrow t[m]\urcorner,\ \Gamma\}\ t[x]} \qquad cov\_induction\_tac\ \ulcorner x\urcorner$$

m will be renamed to a variant, if necessary.

Errors
```
38001 ?0 is not a variable of type ⌜:ℕ⌝
38002 ?0 does not appear free in the conclusion of the goal
38003 ?0 appears free in the assumptions of the goal
```

See Also *induction_tac*

SML
```
val COV_INDUCTION_T : TERM -> (THM -> TACTIC) -> TACTIC;
```

Description This implements course-of-values induction over the natural numbers as a tactical. The term argument must be a variable of type ⌜:ℕ⌝ and must appear free in the conclusion of the goal but not in its assumptions. The inductive hypothesis is passed to the tactic generating function given by the second argument.

Tactic

$$\frac{\{\ \Gamma\ \}\ t[x]}{\begin{array}{c} ttac \\ (\ulcorner\forall m\bullet\ m < x \Rightarrow t\ m\urcorner \\ \vdash \ulcorner\forall m\bullet\ m < x \Rightarrow t\ m\urcorner) \\ (\{\ \Gamma\ \}\ t[x]) \end{array}} \qquad COV\_INDUCTION\_T\ \ulcorner x\urcorner\ ttac$$

Uses Tactic programming.

See Also *cov_induction_tac*, *INDUCTION_T*

Errors

38001 ?0 is not a variable of type ⌜:ℕ⌝
38002 ?0 does not appear free in the conclusion of the goal
38003 ?0 appears free in the assumptions of the goal

SML
```
val induction_tac : TERM -> TACTIC;
```

Description This tactic implements induction over the natural numbers: to prove $t[x]$ it suffices to prove $t[0]$ and to prove $t[x+1]$ on the assumption that $t[x]$. The term argument must be a variable of type ⌜:ℕ⌝ and must appear free in the conclusion of the goal but not in its assumptions.

Tactic

$$\frac{\{\ \Gamma\ \}\ t[x]}{\{\ \Gamma\ \}\ t[0]\ ;\ strip\{t[x],\ \Gamma\}\ t[x{+}1]} \qquad induction\_tac\ \ulcorner x\urcorner$$

See Also *cov_induction_tac*, *INDUCTION_T*

Errors

38001 ?0 is not a variable of type ⌜:ℕ⌝
38002 ?0 does not appear free in the conclusion of the goal
38003 ?0 appears free in the assumptions of the goal

SML
```
val INDUCTION_T : TERM -> (THM -> TACTIC) -> TACTIC;
```

Description This implements induction over the natural numbers as a tactical. The term argument must be a variable of type ⌜:ℕ⌝ and must appear free in the conclusion of the goal but not in its assumptions. The inductive hypothesis is passed to the tactic generating function given by the second argument.

Tactic

$$\frac{\{\ \Gamma\ \}\ t[x]}{\begin{array}{c}\{\ \Gamma\ \}\ t[0]\ ;\ ttac(t[x] \vdash t[x])\\ (\{\ \Gamma\ \}\ t[x{+}1])\end{array}} \qquad INDUCTION\_T\ \ulcorner x \urcorner\ ttac$$

Uses Most commonly used with *asm_tac* to avoid the stripping up of the inductive hypothesis which occurs with *induction_tac*.

See Also *induction_tac*, *COV_INDUCTION_T*

Errors

38001 ?0 is not a variable of type ⌜:ℕ⌝
38002 ?0 does not appear free in the conclusion of the goal
38003 ?0 appears free in the assumptions of the goal

SML
```
val is_ℕ_rep_def : THM;
val ℕ_def : THM;
val div_def : THM;
val ≥_def : THM;
val ≤_def : THM;
val less_def : THM;
val greater_def : THM;
val minus_def : THM;
val mod_def : THM;
val plus_def : THM;
val times_def : THM;
val zero_suc_def : THM;
```

Description These Standard ML variables are bound to the definitions in the theory "ℕ".

SML
```
val plus1_conv : CONV;
```

Description This conversion proves theorems of the form $m = n{+}1$ where m and n are numeric literals.

Tactic

$$\frac{}{\begin{array}{c}\vdash\ {}_{\mathrm{ML}}\ulcorner\ mk\_\mathbb{N}\ m\ \urcorner\\ =\ {}_{\mathrm{ML}}\ulcorner\ mk\_\mathbb{N}\ (m{-}1)\ \urcorner + 1\end{array}} \qquad plus1\_conv\ (mk\_\mathbb{N}\ m)$$

SML
```
val ℕ_cases_thm : THM;
val ¬_≤_thm : THM;
val ¬_less_plus1_thm : THM;
val ¬_less_thm : THM;
val ¬_plus1_≤_thm : THM;
val ¬_plus1_thm : THM;
val cov_induction_thm : THM;
val div_mod_thm : THM;
val div_mod_unique_thm : THM;
val induction_thm : THM;
val ≤_antisym_thm : THM;
val ≤_cases_thm : THM;
val ≤_clauses : THM;
val ≤_plus1_thm : THM;
val ≤_trans_thm : THM;
val ≤_well_order_thm : THM;
val ≤_least_upper_bound_thm : THM;
val less_cases_thm : THM;
val less_clauses : THM;
val less_irrefl_thm : THM;
val less_plus1_thm : THM;
val less_trans_thm : THM;
val less_well_order_thm : THM;
val minimum_¬_thm : THM;
val minus_clauses : THM;
val mod_less_thm : THM;
val one_one_plus1_thm : THM;
val plus1_≤_thm : THM;
val plus1_less_thm : THM;
val plus_assoc_thm : THM;
val plus_assoc_thm1 : THM;
val plus_clauses : THM;
val plus_comm_thm : THM;
val plus_order_thm : THM;
val prim_rec_thm : THM;
val times_comm_thm : THM;
val times_assoc_thm : THM;
val times_plus_distrib_thm : THM;
val times_clauses : THM;
```

Description These Standard ML variables are bound to the theorems in the theory “ℕ”.

9.2.4 Arithmetic Computation

SML
```
signature CompConv = sig
```

Description This is the signature of the structure *CompConv* which contains conversions for performing computations on numeric, string and character literals and derived syntax functions for the arithmetic operations.

SML

val **char_eq_conv** : CONV

Description This conversion proves theorems of the form:

$\vdash b = c \Leftrightarrow t$

Where ⌜b⌝ and ⌜c⌝ are character literals and where ⌜t⌝ is either ⌜T⌝ or ⌜F⌝.

Errors

57200 ?0 is not of the form: ⌜ $_{\mathrm{ML}}$⌜$mk\_char\ b$⌝ = $_{\mathrm{ML}}$⌜$mk\_char\ c$⌝⌝

See Also Proof context *comb*

SML

```
val dest_≤        : TERM -> TERM*TERM
val dest_≥        : TERM -> TERM*TERM
val dest_div      : TERM -> TERM*TERM
val dest_greater  : TERM -> TERM*TERM
val dest_less     : TERM -> TERM*TERM
val dest_minus    : TERM -> TERM*TERM
val dest_mod      : TERM -> TERM*TERM
val dest_plus     : TERM -> TERM*TERM
val dest_times    : TERM -> TERM*TERM
```

Description These are destructor functions for the arithmetic operations.

Errors

57501 ?0 is not of the form: ⌜$x \leq y$⌝
57502 ?0 is not of the form: ⌜$x \geq y$⌝
57503 ?0 is not of the form: ⌜$x\ Div\ y$⌝
57504 ?0 is not of the form: ⌜$x > y$⌝
57505 ?0 is not of the form: ⌜$x < y$⌝
57506 ?0 is not of the form: ⌜$x - y$⌝
57507 ?0 is not of the form: ⌜$x\ Mod\ y$⌝
57508 ?0 is not of the form: ⌜$x + y$⌝
57509 ?0 is not of the form: ⌜$x * y$⌝

SML

```
val is_≤        : TERM -> bool
val is_≥        : TERM -> bool
val is_div      : TERM -> bool
val is_greater  : TERM -> bool
val is_less     : TERM -> bool
val is_minus    : TERM -> bool
val is_mod      : TERM -> bool
val is_plus     : TERM -> bool
val is_times    : TERM -> bool
```

Description These are recogniser functions for the arithmetic operations.

SML

```
val mk_≤        : TERM*TERM -> TERM
val mk_≥        : TERM*TERM -> TERM
val mk_div      : TERM*TERM -> TERM
val mk_greater  : TERM*TERM -> TERM
val mk_less     : TERM*TERM -> TERM
val mk_minus    : TERM*TERM -> TERM
val mk_mod : TERM*TERM -> TERM
val mk_plus : TERM*TERM -> TERM
val mk_times: TERM*TERM -> TERM
```

Description These are constructor functions for the arithmetic operations.

Errors

57510 ?0 is not of type ⌜:ℕ⌝
57511 ?1 is not of type ⌜:ℕ⌝

SML

```
val string_eq_conv : CONV
```

Description These conversions prove theorems of the form:

$\vdash b = c \Leftrightarrow t$

Where ⌜b⌝ and ⌜c⌝ are string literals and where ⌜t⌝ is either ⌜T⌝ or ⌜F⌝.

Errors

57300 ?0 is not of the form: ⌜$_{\mathrm{ML}}$⌜*mk_string b*⌝ = $_{\mathrm{ML}}$⌜*mk_string c*⌝⌝

SML
```
val ≤_conv          : CONV          (* ≤ *)
val ≥_conv          : CONV          (* ≥ *)
val div_conv        : CONV          (* Div *)
val greater_conv    : CONV          (* > *)
val less_conv       : CONV          (* < *)
val minus_conv      : CONV          (* − *)
val mod_conv        : CONV          (* Mod *)
val plus_conv       : CONV          (* + *)
val times_conv      : CONV          (* * *)
```

Description These conversions prove theorems of the form:

$\vdash i \ op \ j = k$

Where ⌜i⌝, ⌜j⌝ and ⌜k⌝ are numeric literals and where *op* is one of the standard arithmetic operators as indicated in the comments above.

(Note that *plus_conv* is the same as the built-in rule *plus_conv* described in DTD009).

Errors

57001 ?0 is not of the form: ⌜$_{ML}$⌜*mk_ℕ m*⌝ ≤ $_{ML}$⌜*mk_ℕ n*⌝⌝
57002 ?0 is not of the form: ⌜$_{ML}$⌜*mk_ℕ m*⌝ ≥ $_{ML}$⌜*mk_ℕ n*⌝⌝
57003 ?0 is not of the form: ⌜$_{ML}$⌜*mk_ℕ m*⌝ *Div* $_{ML}$⌜*mk_ℕ n*⌝⌝
57004 ?0 is not of the form: ⌜$_{ML}$⌜*mk_ℕ m*⌝ > $_{ML}$⌜*mk_ℕ n*⌝⌝
57005 ?0 is not of the form: ⌜$_{ML}$⌜*mk_ℕ m*⌝ < $_{ML}$⌜*mk_ℕ n*⌝⌝
57006 ?0 is not of the form: ⌜$_{ML}$⌜*mk_ℕ m*⌝ − $_{ML}$⌜*mk_ℕ n*⌝⌝
57007 ?0 is not of the form: ⌜$_{ML}$⌜*mk_ℕ m*⌝ *Mod* $_{ML}$⌜*mk_ℕ n*⌝⌝
6085 ?0 is not of the form: ⌜$_{ML}$⌜*mk_ℕ m*⌝ + $_{ML}$⌜*mk_ℕ n*⌝⌝
57009 ?0 is not of the form: ⌜$_{ML}$⌜*mk_ℕ m*⌝ ∗ $_{ML}$⌜*mk_ℕ n*⌝⌝
57010 Cannot compute ?0 (⌜*m − n*⌝ *is undefined when* ⌜*m < n*⌝*)*
57011 Cannot compute ?0 (⌜*m Div n*⌝ *is undefined when* ⌜*n = 0*⌝*)*
57012 Cannot compute ?0 (⌜*m Mod n*⌝ *is undefined when* ⌜*n = 0*⌝*)*

See Also Proof context *comb*

SML
```
val ℕ_eq_conv       : CONV
```

Description This conversion proves theorems of the form:

$\vdash i = j \Leftrightarrow t$

Where ⌜i⌝ and ⌜j⌝ are numeric literals and where ⌜t⌝ is either ⌜T⌝ or ⌜F⌝.

Errors

57100 ?0 is not of the form: ⌜$_{ML}$⌜*mk_ℕ m*⌝ = $_{ML}$⌜*mk_ℕ n*⌝⌝

See Also Proof context *comb*

9.2.5 Linear Arithmetic

SML
```
signature LinearArithmetic = sig
```

Description This is the signature of a structure containing proof procedures for linear arithmetic and related functions.

SML

```
(* Proof Context: lin_arith *)
```

Description This is a proof context whose main purpose is to supply a decision procedure for problems in linear arithmetic.

Contents The rewriting, theorem stripping and conclusion stripping components are as for the proof context *predicates*, q.v, each augmented with conversions effecting the following transformations, where a and b stand for numeric literals and *cncl* refers to an application of one of *ℕ_eq_cancel_conv* or *≤_cancel_conv*

```
a = b        →   ℕ_eq_conv⌜a = b⌝
a ≤ b        →   ≤_conv⌜a ≤ b⌝
¬ t1 = t2    →   cncl⌜t1 + 1 ≤ t2⌝ ∨ cncl⌜t2 + 1 ≤ t1⌝
¬ t1 ≤ t2    →   cncl⌜t2 + 1 ≤ t1⌝
t1 < t2      →   cncl⌜t1 + 1 ≤ t2⌝
t1 ≥ t2      →   cncl⌜t2 ≤ t1⌝
t1 > t2      →   cncl⌜t2 + 1 ≤ t⌝⌝
¬ t1 < t2    →   cncl⌜t2 ≤ t1⌝
¬ t1 ≥ t2    →   cncl⌜t1 + 1 ≤ t2⌝
¬ t1 > t2    →   cncl⌜t1 ≤ t2⌝
t1 = t2      →   cncl⌜t1 = t2⌝
t1 ≤ t2      →   cncl⌜t1 ≤ t2⌝
```

The automatic proof tactic works by *(i)* restripping all the assumptions of the goal, *(ii)* adding the argument theorems to the stock of assumptions using *strip_asm_tac*, *(iii)* applying *contr_tac*, and *(iv)* searching for a linear combination of the assumptions which will reduce, by multiplying out and cancelling like terms, to a contradiction of the form $a = b$ or $a \leq b$ with a and b numeric literals. The automatic proof conversion just tries to prove its argument, t say, using the automatic proof tactic and returns $t = T$ if it succeeds.

Other components of the proof context are as for *predicates*.

For example, *PC_T1"lin_arith"prove_tac*[] will prove any of the following goals:

```
    ([], ⌜∀a b c•a ≤ b ∧ (a + b < c + a) ⇒ a < c⌝)
    ([], ⌜∀a b c•a ≥ b ∧ ¬ b < c ⇒ a ≥ c⌝)
    ([], ⌜∀a b c•a + 2*b < 2*a ⇒ b + b < a⌝)
    ([], ⌜∀ x y• ¬ (2*x + y = 4 ∧ 4*x + 2*y = 7)⌝)
```

In cases where the automatic proof tactic cannot prove the goal, it is sometimes possible to use *scale_asm_tac*, q.v., or one of its variants to transform the goal into a form it can prove.

See Also *'lin_arith*, *scale_asm_tac*

SML

```
(* Proof Context: 'lin_arith *)
```

Description This is a component proof context whose main purpose is to supply a decision procedure for problems in linear arithmetic.

Contents The rewriting, theorem stripping and conclusion stripping components are as for the proof context *lin_arith* but without any of the material from the proof context *predicates*. The automatic proof components are as for *lin_arith*. Other components are blank.

A typical use of the proof context would be to solve problems containing a mixture of (linear) arithmetic and set theory. For example, *MERGE_PCS_T1*["*sets_ext*", "'*lin_arith*"]*prove_tac*[]will prove any of the following goals:

$$([], \ulcorner \forall m\bullet \{i \mid m \le i \land i < m+3\} = \{m;\ m+1;\ m+2\}\urcorner)$$
$$([], \ulcorner \{(i, j) \mid 30*i = 105*j\} = \{(i, j) \mid 2*i = 7*j\}\urcorner)$$
$$([], \ulcorner \{i \mid 5*i = 6*i\} = \{0\}\urcorner)$$

The automatic proof procedures may give rise to the following error message (in which the message from the function which first detected the problem is given in brackets):

Errors

82200 Could not prove theorem with conclusion ?1 (?0)

SML

```
structure LinearArithmeticTools : sig
        type POLY    (*= (INTEGER * int) list *);
        datatype CONSTRAINT_TYPE = Eq | LessEq | Less;
        type CONSTRAINT (* = POLY * CONSTRAINT_TYPE * INTEGER * POLY *);
        val normalise_poly : POLY -> POLY;
        val normalise_constraint : CONSTRAINT -> CONSTRAINT;
        val format_poly : POLY -> string;
        val format_constraint : CONSTRAINT -> string;
        val mult_and_add_poly : INTEGER -> POLY -> INTEGER -> POLY -> POLY;
        val mult_and_add_constraint
                : INTEGER -> CONSTRAINT -> INTEGER -> CONSTRAINT -> CONSTRAINT;
        val lin_arith_contr : CONSTRAINT list -> POLY;
end;
```

Description This is a structure contained in the structure *LinearArithmetic* giving types and functions used to represent systems of linear constraints with integer coefficients over the rational numbers. Most importantly, *lin_arith_contr* provides a decision procedure for the unsatisfiability of such systems. In more detail:

Degree one polynomials in many variables x_1, x_2, ... (henceforth, just *polynomials*) are represented, using the type *POLY*, as lists of pairs of integers, a pair (a, i) indicating that the i-th variable has coefficient a. The function *normalise_poly* puts any polynomial in a normal form, in which the variables in such a polynomial occur in strictly increasing order and pairs with zero coefficients are omitted. Provided two polynomials, p and q, are in normal form, *mult_and_add_poly i p j q* computes the normalised representation of $i*p + j*q$. N.B. for efficiency reasons, these functions do not check that their arguments are normalised and may produce results which are not only not normalised but also incorrect in such cases.

Constraints are labelled with an indicator of their origin. This is a polynomial, and typically one would label the j-th constraint with x_j. The provenance of a generated constraint is then represented using a polynomial, where e.g. $2x_1+3x_2$ means add twice the polynomial labelled with x_1 to thrice that labelled x_2. A constraint $a_1*x_1 + a_2*x_2 + ... a_k * x_k\ R\ c$ with provenance p is then represented as a 4-tuple $([(a_1,\ x_1),\ (a_2,\ x_2),\ ...],\ R,\ c,\ p)$. The polynomials in a constraint may be put into the normal form described above using *normalise_constraint* and constraints may be multiplied by constants and added using *mult_and_add_constraint*.

Given a list, *cs*, of constraints (not necessarily normalised), *lin_arith_contr cs* determines whether *cs* is unsatisfiable (i.e. contradictory): if so, it returns a normalised polynomial indicating, in terms of of the origin labels in the inputs, a linear combination of the inputs which reduces, by cancellation, to an unsatisfiable constraint with no variables; if not, it fails with error number 82110.

Errors

82110 System is satisfiable

SML

```
val lin_arith_rule : TERM list -> THM;
val lin_arith_rule1 : TERM list -> THM;
```

Description Given a system, $\Gamma = [r_1, r_2, ...]$, of numeric constraints, r_i, of the form $(s_i{:}\mathbb{N}) = t_i$or $s_i \leq t_i$ these rules attempt to prove a theorem of the form $\Gamma \vdash F$ (terms in Γ which are not of either of these forms are ignored and do not appear in the assumptions of the result theorem). The usual interface to these rules is via the decision procedures in the proof context *lin_arith*, q.v., e.g. as invoked by *PC_T1*"*lin_arith*"*prove_tac*[]. The following description of the detailed operation of the rule is mainly intended for those who wish to produce similar proof procedures for other number systems. The rules operate as if by the following steps:

1. The system of constraints is replaced by a new system in which the s_i and $t_i$are formed only from numeric literals, variables, + and ∗. This is done by choosing variables, x_k, such that each r_i is a substitution instance of a constraint of the form $u_{i1} + u_{i2} + ... = v_{i1} + v_{i2} + ...$ or $u_{i1} + u_{i2} + ... \leq v_{i1} + v_{i2} + ...$, where the summands u_{ij} and v_{ij} have one of the forms: $a_{ij}{*}x_k$, a_{ij} or x_k where $a_{ij}$is a numeric literal; this is done using as few variables as possible subject to the constraint that a u_{ij} or a v_{ij} will only have the form x_k if it corresponds to a summand in r_i which is not of the form $a{*}t$ where a is a numeric literal.

2. In the case of *lin_arith_rule1* only, the new system is now augmented with an additional inequality $x_k \geq 0$ for each variable x_k. (These additional inequalities do not appear in the assumptions of the result theorem.)

3. An attempt to derive a contradiction from the new system, $\Delta = [r'_1, r'_2, ...]$ say, is then made using the search procedure *lin_arith_contr*. If this attempt fails then an exception is raised (see below), otherwise, it produces integer values, $i_1, i_2, ...$ say, such that the linear combination $i_1{*}r'_1 + i_2{*}r'_2 + ...$ of the constraints r'_i reduces by multiplying through and cancelling like terms to a term of the fom $a = b$ (resp. $a \leq b$), where a and b are numeric literals such that $\neg a = b$ (resp. $\neg a \leq b$). (Multiplying out by a negative value, $-i$ say, amounts to interchanging the left- and right-hand side of the constraint, replacing $\leq$ by $\geq$ if appropriate, then multiplying by i.)

4. The theorem $\Delta \vdash F$ is then derived using *anf_conv* to effect the multiplication and cancelling, and then $\Gamma \vdash F$ is derived from it using the substitution constructed in step 1.

If no contradiction can be derived then the rules both raise an exception which, if not handled, causes a diagnostic print of the input terms and of the system of linear equations which was derived from them. This is done by calling *fail*, q.v., with its third argument set to a list containing a function which produces the diagnostic print-out by side-effect. The function *get_message*, or *get_message_text* should not therefore be evaluated on the exception value unless the diagnostics are to be printed immediately.

Errors

82110	*System is satisfiable*
82111	*A system with no constraints is satisfiable*
82112	*No constraints of the form* $\vdash t1 \leq t2$ *or* $\vdash (t1{:}\mathbb{N}) = t2$ *could be derived from the assumptions*

SML

```
val lin_arith_tac : TACTIC;
val lin_arith_tac1 : TACTIC;
```

Description These tactics are primarily intended for use in conjunction with the linear arithmetic proof context *lin_arith* and *'lin_arith*, q.v. The normal interface to the tactics is via the decision procedures in these proof contexts, as in for example: *PC_T1"lin_arith"prove_tac*[].

The tactics do however, have possible applications in specialised tactic programming, in which circumstances their behaviour may be understood from their definition, in terms of *lin_arith_rule* or *lin_arith_rule1*, q.v., essentially as:

```
val lin_arith_tac = GET_ASMS_T (f_thm_tac o lin_arith_rule o map concl);
val lin_arith_tac1 = GET_ASMS_T (f_thm_tac o lin_arith_rule1 o map concl);
```

Uses The most likely application is in specialised tactic programming in situations where it is known that the assumptions are already in the normal form produced by the proof context *lin_arith* and it is important for performance not to restrip them.

SML

```
val make_≤_conv : CONV
```

Description This conversion transforms arithmetic relations formed with $<$, $\geq$ or $>$, or the negation of one formed with $=$, $\leq$, $<$, $\geq$ or $>$ into relations involving only $\leq$ as follows:

```
t1 < t2       →   t1 + 1 ≤ t2
t1 ≥ t2       →   t2 ≤ t1
t1 > t2       →   t2 + 1 ≤ t1
¬ t1 = t2     →   t1 + 1 ≤ t2 ∨ t2 + 1 ≤ t1
¬ t1 ≤ t2     →   t2 + 1 ≤ t1
¬ t1 < t2     →   t2 ≤ t1
¬ t1 ≥ t2     →   t1 + 1 ≤ t2
¬ t1 > t2     →   t1 ≤ t2
```

Uses The conversion is intended for use in tactic and conversion programming. The normal interactive interface is via rewriting or stripping in the proof context *lin_arith*, which performs other useful simplifications. For example, the call

ProofPower Input

```
PC_C1"lin_arith"rewrite_conv[]⌜¬x + 2*y + x + 3 = y + 2 + 2*x + y⌝;
```

produces the following output:

ProofPower Output

```
val it = ⊢ ¬ x + 2 * y + x + 3 = y + 2 + 2 * x + y ⇔ T : THM
```

Errors

```
82012  ?0 is not of the form ⊢ (t1:ℕ) R1 t2 or ⊢ ¬(t1:ℕ) R2 t2 where R2 is one
       of <, ≥ or > and R2 is one of =, ≤, <, ≥ or >
```

SML

```
val scale_asm_tac : TERM -> TERM -> TACTIC;
val scale_nth_asm_tac : TERM -> int -> TACTIC;
val list_scale_nth_asm_tac : (TERM * int) list -> TACTIC;
```

Description *scale_asm_tac t asm* removes *asm* from the assumption list and strips in a new assumption obtained by "scaling *asm* by the factor *t*" as explained in the description of *scale_rule*.

scale_nth_asm_tac is just like *scale_asm_tac* but allows the assumption to be identified by its number, and *list_scale_nth_asm_tac* is a list analogue of *scale_nth_asm_tac*, allowing several assumptions to be scaled at once (or indeed allowing one or more assumptions to be scaled by different factors).

For example, *scale_asm_tac*⌜(1+x)⌝⌜y+1 ≤ z + 1⌝ removes the assumption ⌜y+1 ≤ z + 1⌝ from the assumption list and strips into the assumptions the new assumption: ⌜(1 + x)∗(y + 1) ≤ (1 + x)∗(z + 1)⌝. The scaling tactics are primarily intended for use with the decision procedure in the proof context *lin_arith* or other proof contexts containing *'lin_arith*, to adjust a problem which it cannot solve into one it can. For example, consider the goal:

```
([], ⌜a∗(d+1) = c∗(b+1) ∧ c∗(f+1) = e∗(d+1) ⇒ a∗(f+1) = e∗(b+1)⌝)
```

To prove this, we first use *contr_tac*(say in the proof context *hol*) to move everything into the assumptions, giving:

```
(∗ 3 ∗) ⌜a ∗ (d + 1) = c ∗ (b + 1)⌝
(∗ 2 ∗) ⌜c ∗ (f + 1) = e ∗ (d + 1)⌝
(∗ 1 ∗) ⌜¬ a ∗ (f + 1) = e ∗ (b + 1)⌝
```

(If this were done in the proof context *lin_arith*, we would get an unwanted case split at this point.) We now see that, if we multiply assumption *1* by $d + 1$, assumption *2* by $b + 1$ and assumption *3* by $f + 1$, then a number of like terms will appear which the linear arithmetic decision procedure should be able to take advantage of, and indeed the goal may be solved by the following tactic:

```
        list_scale_nth_asm_tac[(⌜d+1⌝, 1), (⌜b+1⌝, 2), (⌜f+1⌝, 3)]
THEN PC_T1"lin_arith"asm_prove_tac[]
```

N.B. this tactic strengthens the goal in general, i.e. it may lead to unprovable subgoals even when the original goal was provable (since the scale factor may not be provably non-zero).

Errors as for *scale_rule*and *DROP_ASM_T* and its variants

SML

val **scale_rule** : *TERM −> THM −> THM*

Description In the simplest cases, given a term, t, of type ⌜:ℕ⌝ and a theorem with conclusion of the form $(t1 : \mathbb{N}) = t2$ or $t1 \leq t2$, *scale_rule* returns a theorem which expresses the result of multiplying both sides of the conclusion of the theorem by t. More generally, *scale_rule t* processes theorems whose conclusions are atomic arithmetic propositions or negations of same as follows:

$t1 = t2$	$\rightarrow$	$t * t1 = t * t2$
$t1 \leq t2$	$\rightarrow$	$t * t1 \leq t * t2$
$t1 < t2$	$\rightarrow$	$t * t1 + t \leq t * t2$
$t1 \geq t2$	$\rightarrow$	$t * t2 \leq t * t1$
$t1 > t2$	$\rightarrow$	$t * t2 + t \leq t * t1$
$\neg t1 = t2$	$\rightarrow$	$t * t1 + t \leq t * t2 \lor t * t2 + t \leq t * t1$
$\neg t1 \leq t2$	$\rightarrow$	$t * t2 + t \leq t * t1$
$\neg t1 < t2$	$\rightarrow$	$t * t2 \leq t * t1$
$\neg t1 \geq t2$	$\rightarrow$	$t * t1 + t \leq t * t2$
$\neg t1 > t2$	$\rightarrow$	$t * t1 \leq t * t2$

Thus in the first two cases *scale_rule t* just scales the equation or inequality in the theorem by the factor t and in the other cases the theorem is first converted to one or more inequalities with $\leq$ and then scaled by t.

Errors

82010 ?0 does not have type ⌜:ℕ⌝
82011 ?0 is not of the form $\vdash (t1{:}\mathbb{N})\ R\ t2$ *or* $\vdash \neg(t1{:}\mathbb{N})\ R\ t2$ *where R is one of* $=, \leq, <, \geq$ *or* $>$

See Also *scale_asm_tac*

SML

```
val ℕ_eq_cancel_conv : CONV
val ≤_cancel_conv : CONV
```

Description *ℕ_eq_cancel_conv* (resp. *≤_cancel_conv*) normalises arithmetic equations (resp. inequalities formed with ≤) by putting the left- and right-hand sides in additive normal form, in the sense of *anf_conv*, q.v., then cancelling like terms.

For example, the calls:

ProofPower Input

```
ℕ_eq_cancel_conv⌜x + 2*y + 5*z + 3 = 1 +6*y + z⌝;
≤_cancel_conv⌜x + 2*y + x + 3 ≤ y + 2 + 2*x + y⌝;
```

produce the following output

ProofPower Output

```
val it = ⊢ x + 2 * y + 5 * z + 3 = 1 + 6 * y + z ⇔ 2 + x + 4 * z = 4 * y : THM
val it = ⊢ x + 2 * y + x + 3 ≤ y + 2 + 2 * x + y ⇔ 1 ≤ 0 : THM
```

Note that, as in the last example, if the result of the normalisation has numeric literals on both left- and right-hand sides, then one of the literals will be *0*. However, the truth value is not evaluated. *ℕ_eq_conv* or *≤_conv* may be used to perform the evaluation, where required.

Uses The conversions are intended for use in tactic and conversion programming. The normal interactive interface is via rewriting or stripping in the proof context *lin_arith*, which performs other useful simplifications. For example, the call

ProofPower Input

```
PC_C1"lin_arith"rewrite_conv[]⌜x + 2*y + x + 3 ≤ y + 2 + 2*x + y⌝;
```

produces the following output:

ProofPower Output

```
val it = ⊢ x + 2 * y + x + 3 ≤ y + 2 + 2 * x + y ⇔ F : THM
```

Errors

```
82120 ?0 is not of the form (t1:ℕ) = t2 or is already in normal form
82121 ?0 is not of the form (t1:ℕ) ≤ t2 or is already in normal form
```

See Also *lin_arith*, *ℕ_eq_conv*, *≤_conv*, *make_≤_conv*

9.2.6 Integers

SML

```
signature ℤ = sig
```

Description This provides the basic definitions proof support for the HOL theory of integers. It creates the theory ℤ.

SML

```
(* Proof Context: 'ℤ *)
```

Description A component proof context for handling the basic arithmetic operations for Z.

Expressions and predicates treated by this proof context are constructs formed from:

```
+, ∗, −, abs, div, mod, ℤ, ≤, <, ≥, >, =, ℕ
```

Contents

Rewriting:

```
ℤ_plus_conv, ℤ_times_conv, ℤ_subtract_minus_conv
ℤ_abs_conv, ℤ_div_conv, ℤ_mod_conv
ℤ_eq_conv, ℤ_≤_conv, ℤ_less_conv
ℤ_≥_≤_conv, ℤ_greater_less_conv,
ℤ_plus_clauses, ℤ_minus_clauses, ℤ_≤_clauses
ℤ_less_clauses, ℤ_¬_≤_thm, ℤ_¬_less_thm
```

Stripping theorems:

```
ℤ_eq_conv, ℤ_≤_conv, ℤ_less_conv
ℤ_≥_≤_conv, ℤ_greater_less_conv,
ℤ_plus_clauses, ℤ_minus_clauses, ℤ_≤_clauses
ℤ_less_clauses,
and all the above pushed through ¬

ℤ_¬_≤_thm, ℤ_¬_less_thm, ℤ_≤_conv, ℤ_less_conv
```

Stripping conclusions: as for stripping theorems.

Rewriting canonicalisation: blank.

Automatic proof procedures: *basic_prove_tac*, *basic_prove_conv*. Automatic existence prover: blank.

SML

```
val dest_ℤ_≤ : TERM -> TERM * TERM;
val dest_ℤ_≥ : TERM -> TERM * TERM;
val dest_ℤ_abs : TERM -> TERM;
val dest_ℤ_div : TERM -> TERM * TERM;
val dest_ℤ_greater : TERM -> TERM * TERM;
val dest_ℤ_less : TERM -> TERM * TERM;
val dest_ℤ_minus : TERM -> TERM;
val dest_ℤ_mod : TERM -> TERM * TERM;
val dest_ℤ_plus : TERM -> TERM * TERM;
val dest_ℤ_signed_int : TERM -> INTEGER;
val dest_ℤ_subtract : TERM -> TERM * TERM;
val dest_ℤ_times : TERM -> TERM * TERM;
```

Description These are derived destructor functions for the basic arithmetic operations on the integers. An optionally signed integer literal, *signed_int*, is taken to be either a numeric literal or the result of applying (∼_) to a numeric literal. The other constructors correspond directly to the arithmetic operations of the theory ℤ with alphabetic names assigned to give valid ML name as needed (*greater* :<, *less* :>, *minus*: ∼, *plus* : +, *subtract* : −, *times* : ∗).

As usual, there are also corresponding discriminator (*is_ ...*) and constructor functions (*dest_ ...*). For programming convenience, *dest_ℤ_signed_int* returns *0* and *mk_ℤ_signed_int* returns *true* when applied to ∼*0*, but *mk_ℤ_signed_int* cannot be used to construct such a term.

Errors

```
93101 ?0 is not of the form ⌜i ≤Z j⌝
93102 ?0 is not of the form ⌜i ≥Z j⌝
93103 ?0 is not of the form ⌜AbsZ i⌝
93104 ?0 is not of the form ⌜i DivZ j⌝
93105 ?0 is not of the form ⌜i >Z j⌝
93106 ?0 is not of the form ⌜i <Z j⌝
93107 ?0 is not of the form ⌜∼Zi⌝
93108 ?0 is not of the form ⌜i ModZ j⌝
93109 ?0 is not of the form ⌜i +Z j⌝
93110 ?0 is not an optionally signed integer literal (theory ℤ)
93111 ?0 is not of the form ⌜i −Z j⌝
93112 ?0 is not of the form ⌜i ∗Z j⌝
```

SML

```
val Is_ℤ_Rep_def : THM;            val ℤ_ℕℤ_def : THM;
val ℤ_minus_def : THM;             val ℤ_plus_def : THM;
val ℤ_subtract_def : THM;          val ℤ_times_def : THM;
val ℤ_≤_def : THM;                 val ℤ_less_def : THM;
val ℤ_≥_def : THM;                 val ℤ_greater_def : THM;
val ℤ_abs_def : THM;                   val ℤ_mod_def : THM;
val ℤ_div_def : THM;
```

Description These are the ML bindings of the definitions of the theory ℤ.

SML

```
val is_ℤ_≤ : TERM -> bool;
val is_ℤ_≥ : TERM -> bool;
val is_ℤ_abs : TERM -> bool;
val is_ℤ_div : TERM -> bool;
val is_ℤ_greater : TERM -> bool;
val is_ℤ_less : TERM -> bool;
val is_ℤ_minus : TERM -> bool;
val is_ℤ_mod : TERM -> bool;
val is_ℤ_plus : TERM -> bool;
val is_ℤ_signed_int : TERM -> bool;
val is_ℤ_subtract : TERM -> bool;
val is_ℤ_times : TERM -> bool;
```

Description These are derived discriminator functions for the Z basic arithmetic operations. See the documentation for the destructor functions (*dest_ℤ_plus* etc.) for more information.

SML

```
val mk_ℤ_≤ : TERM * TERM -> TERM;
val mk_ℤ_≥ : TERM * TERM -> TERM;
val mk_ℤ_abs : TERM -> TERM;
val mk_ℤ_div : TERM * TERM -> TERM;
val mk_ℤ_greater : TERM * TERM -> TERM;
val mk_ℤ_less : TERM * TERM -> TERM;
val mk_ℤ_minus : TERM -> TERM;
val mk_ℤ_mod : TERM * TERM -> TERM;
val mk_ℤ_plus : TERM * TERM -> TERM;
val mk_ℤ_signed_int : INTEGER -> TERM;
val mk_ℤ_subtract : TERM * TERM -> TERM;
val mk_ℤ_times : TERM * TERM -> TERM;
```

Description These are derived constructor functions for the Z basic arithmetic operations. See the documentation for the destructor functions (*dest_ℤ_plus* etc.) for more information.

Errors

```
93201 ?0 does not have type ℤ
```

SML

```
val ℕℤ_plus_homomorphism_thm : THM;
val ℕℤ_plus_homomorphism_thm1 : THM;
val ℕℤ_times_homomorphism_thm : THM;
val ℕℤ_times_homomorphism_thm1 : THM;
val ℕℤ_≤_thm : THM;                      val ℕℤ_less_thm : THM;
val ℕℤ_one_one_thm : THM;                val ℤ_¬_≤_thm : THM;
val ℤ_¬_ℕ_thm : THM;                     val ℤ_¬_less_thm : THM;
val ℤ_≤_≤_0_thm1 : THM;                  val ℤ_≤_≤_0_thm : THM;
val ℤ_≤_antisym_thm : THM;               val ℤ_≤_cases_thm : THM;
val ℤ_≤_clauses : THM;                   val ℤ_≤_less_eq_thm : THM;
val ℤ_≤_minus_thm : THM;                 val ℤ_≤_plus_ℕ_thm : THM;
val ℤ_≤_refl_thm : THM;                  val ℤ_≤_trans_thm : THM;
val ℤ_∈_ℕ_thm : THM;                     val ℤ_ℕ_¬_minus_thm : THM;
val ℤ_ℕ_¬_plus1_thm : THM;               val ℤ_ℕ_abs_minus_thm : THM;
val ℤ_ℕ_abs_thm : THM;                   val ℤ_ℕ_cases_thm : THM;
val ℤ_ℕ_induction_thm : THM;             val ℤ_ℕ_plus_thm : THM;
val ℤ_ℕ_plus1_thm : THM;                 val ℤ_ℕ_times_thm : THM;
val ℤ_abs_ℕ_thm : THM;                   val ℤ_abs_eq_0_thm : THM;
val ℤ_abs_minus_thm : THM;               val ℤ_abs_thm : THM;
val ℤ_abs_times_thm : THM;               val ℤ_abs_plus_thm : THM;
val ℤ_cases_thm : THM;                   val ℤ_cases_thm1 : THM;
val ℤ_div_mod_unique_thm : THM;          val ℤ_eq_thm : THM;
val ℤ_eq_thm1 : THM;                     val ℤ_induction_thm : THM;
val ℤ_≤_induction_thm : THM;             val ℤ_cov_induction_thm : THM;
val ℤ_fun_∃_thm : THM;                   val ℤ_less_≤_trans_thm : THM;
val ℤ_less_clauses : THM;                val ℤ_less_cases_thm : THM;
val ℤ_less_irrefl_thm : THM;             val ℤ_less_less_0_thm : THM;
val ℤ_less_less_0_thm1 : THM;            val ℤ_less_trans_thm : THM;
val ℤ_minus_≤_thm : THM;                 val ℤ_minus_ℕ_≤_thm : THM;
val ℤ_minus_clauses : THM;               val ℤ_minus_less_thm : THM;
val ℤ_minus_thm : THM;                   val ℤ_plus_assoc_thm : THM;
val ℤ_plus_assoc_thm1 : THM;             val ℤ_plus_clauses : THM;
val ℤ_plus_comm_thm : THM;               val ℤ_plus_eq_thm : THM;
val ℤ_plus_minus_thm : THM;              val ℤ_plus_order_thm : THM;
val ℤ_plus0_thm : THM;                   val ℤ_times_assoc_thm : THM;
val ℤ_times_assoc_thm1 : THM;            val ℤ_times_clauses : THM;
val ℤ_times_comm_thm : THM;              val ℤ_times_eq_0_thm : THM;
val ℤ_times_minus_thm : THM;             val ℤ_times_order_thm : THM;
val ℤ_times_plus_distrib_thm : THM;      val ℤ_times0_thm : THM;
val ℤ_times1_thm : THM;
```

Description These are the ML value bindings for the theorems saved in the theory ℤ.

SML

$val\ \mathbb{Z}$**_cov_induction_tac** : $TERM\ \text{->}\ TACTIC$

Description This tactic implements course-of-values induction over subsets of the integers that are bounded below: to prove that $t[x]$holds whenever $x \geq j$, it suffices to prove $t[i/x]$ on the assumptions that $j{\leq}i$ and $\forall k \bullet j \leq k \land k < i \Rightarrow t[k/x]$

The term argument must be a variable that appears free in the conclusion of the goal. The variable must also appear free once, and only once, in the assumptions, in an assumption of the form $j{\leq}x$.

Tactic

$$\frac{\{\ \Gamma,\ j{\leq}x\}\ t[x]}{strip\ \{j{\leq}x,\ {}^{\ulcorner}_{Z}\forall k\bullet\ j{\leq}k \land k{<}x \Rightarrow t[k]^{\urcorner},\ \Gamma\}\ t[x]} \quad \mathbb{Z}\_{\leq}\_induction\_tac\ {}^{\ulcorner}_{Z}x^{\urcorner}$$

See Also $\mathbb{Z}\_cases\_thm$, $intro\_\forall\_tac$, $\mathbb{Z}\_induction\_tac$, $\mathbb{Z}\_\mathbb{N}\_induction\_tac$, $\mathbb{Z}\_{\leq}\_induction\_tac$

Errors

93401 ?0 is not a variable of type $^{\ulcorner}{:}\mathbb{Z}^{\urcorner}$
93402 A term of the form ${}^{\ulcorner}_{Z}j \leq i^{\urcorner}$ *where i is the induction variable could not be found in the assumptions*
93403 ?0 appears free in more than one assumption of the goal
93404 ?0 does not appear free in the conclusions of the goal

SML

$val\ \mathbb{Z}$**_induction_tac** : $TERM\ \text{->}\ TACTIC$

Description An induction-like tactic for the integers, based on the fact that any subset of the integers containing *1* and closed under negation and addition must contain every integer.

Tactic

$$\frac{\{\ \Gamma\ \}\ t}{\begin{array}{c}\{\ \Gamma\ \}\ t[1/x]\ ;\\ strip\{t[i/x],\ \Gamma\}\ t[\sim i/x]\ ;\\ strip\{t[i/x] \land t[j/x],\ \Gamma\}\ t[i{+}j/x]\end{array}} \quad \mathbb{Z}\_\mathbb{N}\_induction\_tac\ {}^{\ulcorner}x^{\urcorner}$$

See Also $\mathbb{Z}\_\mathbb{N}\_induction\_tac$, $\mathbb{Z}\_intro\_\forall\_tac$, $\mathbb{Z}\_cases\_thm$

Errors As for *gen_induction_tac*.

SML

$val\ \mathbb{Z}\_\leq\_$**induction_tac** : $TERM$ −> $TACTIC$

Description This tactic implements induction over subsets of the integers that are bounded below: to prove that $t[x]$holds whenever $x \geq j$, it suffices to prove $t[j/x]$ and to prove $t[x+1/x]$ on the assumption that $t[x]$ and $x \geq j$.

The term argument must be a variable that appears free in the conclusion of the goal. The variable must also appear free once, and only once, in the assumptions, in an assumption of the form $j \geq x$.

Tactic

$$\frac{\{\ \Gamma;\ j\ \leq\ x\ \}\ t}{\{\ \Gamma\ \}\ t[j/x]\ ;\ strip\{t,\ j\ \leq\ x,\ \Gamma\}\ t[x{+}1/x]} \qquad \mathbb{Z}\_\leq\_induction\_tac\ \ulcorner x\urcorner$$

See Also $\mathbb{Z}\_cases\_thm$, $intro\_\forall\_tac$, $\mathbb{Z}\_induction\_tac$, $\mathbb{Z}\_\mathbb{N}\_induction\_tac$, $\mathbb{Z}\_cov\_induction\_tac$

Errors

93401 ?0 is not a variable of type $\ulcorner:\mathbb{Z}\urcorner$
93402 A term of the form $\ulcorner_{\mathbb{Z}} j \leq i\urcorner$ *where i is the induction variable could not be found in the assumptions*
93403 ?0 appears free in more than one assumption of the goal
93404 ?0 does not appear free in the conclusions of the goal

SML

$val\ \mathbb{Z}\_\mathbb{N}\_$**induction_tac** : $TACTIC$

Description This tactic implements induction over the natural numbers (as a subset of the HOL integers): to prove $\mathbb{N}\mathbb{Z}\ 0 \leq x \Rightarrow t$, it suffices to prove $t[0/x]$ and to prove $t[x+1/x]$ on the assumption that t. The conclusion of the goal must have the form $\mathbb{N}\mathbb{Z}\ 0 \leq x \Rightarrow t$.

Tactic

$$\frac{\{\ \Gamma\ \}\ \mathbb{N}\mathbb{Z}\ 0\ \leq\ x\ \Rightarrow\ t}{\{\ \Gamma\ \}\ t[0/x]\ ;\ strip\{t,\ \Gamma\}\ t[x{+}1/x]} \qquad \mathbb{Z}\_\mathbb{N}\_induction\_tac$$

See Also $\mathbb{Z}\_cases\_thm$, $intro\_\forall\_tac$, $\mathbb{Z}\_induction\_tac$, $\mathbb{Z}\_\leq\_induction\_tac$, $\mathbb{Z}\_cov\_induction\_tac$

Errors As for *gen_induction_tac1*.

SML

val $\mathbb{Z}$_$\mathbb{N}$_**plus_conv** : *CONV*; *val* $\mathbb{Z}$_$\mathbb{N}$_**times_conv** : *CONV*;

val $\mathbb{Z}$_**subtract_minus_conv** : *CONV*; *val* $\mathbb{Z}$_**greater_less_conv** : *CONV*
val $\mathbb{Z}$_$\geq$_$\leq$_**conv** : *CONV*;

val $\mathbb{Z}$_**plus_conv** : *CONV*; *val* $\mathbb{Z}$_**times_conv** : *CONV*
val $\mathbb{Z}$_**abs_conv** : *CONV*; *val* $\mathbb{Z}$_**div_conv** : *CONV*
val $\mathbb{Z}$_**mod_conv** : *CONV*; *val* $\mathbb{Z}$_$\leq$_**conv** : *CONV*
val $\mathbb{Z}$_**less_conv** : *CONV* *val* $\mathbb{Z}$_**eq_conv** : *CONV*

Description These conversions are used to perform evaluation of arithmetic expressions involving numeric literal operands. The normal interface to the conversion is via the proof context $\mathbb{Z}$ and other proof contexts which contain it.

The first block above gives conversions to evaluate expressions of the form $i\ op\ j$ where i and j are numeric literals and op is one of $+$ or $*$. The second block gives conversions to transform terms of the form $i - j$, $i > j$, $i > j$ and $i \in \mathbb{N}$ into $i + \sim j$, $j < i$, $j \leq i$ and $0 \leq i$ respectively. The third block give conversions which evaluate expressions of the form $i\ op\ j$ or $abs\ i$, where op is one of $+$, $*$, div, mod, $\leq$, $<$, or $=$, and where i and j are signed integer literals (i.e., either numeric literals or of the form $\sim k$, where k is a numeric literal). Thus the second block of conversions transform expressions of the form $i - j$, $i > j$, $i \geq j$ and $i \in \mathbb{N}$ into a form which can be evaluated by the conversions in the third block if i and j are signed literals.

Errors

93301 ?0 is not of the form ?1 where ⌜i⌝ and ⌜j⌝ are numeric literals (theory $\mathbb{Z}$)
93302 ?0 is not of the form ?1 (theory $\mathbb{Z}$)
93303 ?0 is not of the form ?1 where ⌜i⌝ and ⌜j⌝ are optionally signed numeric literals (theory $\mathbb{Z}$)

9.2.7 Integer Proof Support

SML

signature $\mathbb{Z}$**ArithmeticTools** = *sig*

Description This is the signature of a structure containing arithmetic and an automatic linear arithmetic prover for the HOL integers (as defined in the theory $\mathbb{Z}$).

SML

```
(* Proof Context: ℤ_lin_arith *)
```

Description This is a proof context whose main purpose is to supply a decision procedure for problems in linear arithmetic over the HOL integers.

Contents The rewriting, theorem stripping and conclusion stripping components are as for the proof context *predicates*, q.v, each augmented with conversions effecting the following transformations, where a and b stand for numeric literals.

$a = b$	$\rightarrow$	$ℤ\_eq\_conv⌜a = b⌝$
$a \leq b$	$\rightarrow$	$ℤ\_\leq\_conv⌜a \leq b⌝$
$t1 = t2$	$\rightarrow$	$ℤ\_eq\_cancel\_conv⌜t1 = t2⌝$
$t1 \leq t2$	$\rightarrow$	$ℤ\_\leq\_cancel\_conv⌜t1 \leq t2⌝$
$\neg\ t1 = t2$	$\rightarrow$	$t1 < t2 \vee t2 < t1$
$\neg\ t1 \leq t2$	$\rightarrow$	$t2 < t1$
$t1 < t2$	$\rightarrow$	$t1 + 1 \leq t2$
$t1 \geq t2$	$\rightarrow$	$t2 \leq t1$
$t1 > t2$	$\rightarrow$	$t2 < t$
$\neg\ t1 < t2$	$\rightarrow$	$t2 \leq t1$
$\neg\ t1 \geq t2$	$\rightarrow$	$\neg\ t2 \leq t1$
$\neg\ t1 > t2$	$\rightarrow$	$t1 \leq t2$
$t1 = t2$	$\rightarrow$	$t1 = t2$
$t1 \leq t2$	$\rightarrow$	$t1 \leq t2$

(where all variables are of type ℤ).

The automatic proof tactic works by *(i)* restripping all the assumptions of the goal, *(ii)* adding the argument theorems to the stock of assumptions using *strip_asm_tac*, *(iii)* applying *contr_tac*, and *(iv)* searching for a linear combination of the assumptions which will reduce, by multiplying out and cancelling like terms, to a contradiction of the form $a = b$ or $a \leq b$ with a and b numeric literals. The automatic proof conversion just tries to prove its argument, t say, using the automatic proof tactic and returns $t = T$ if it succeeds.

Other components of the proof context are as for *predicates*.

For example, *PC_T1*"*ℤ_lin_arith*"*prove_tac*[] will prove any of the following goals:

```
([], ⌜∀a b c:ℤ•a ≤ b ∧ (a + b < c + a) ⇒ a < c⌝)
([], ⌜∀a b c:ℤ•a ≥ b ∧ ¬ b < c ⇒ a ≥ c⌝)
([], ⌜∀a b c:ℤ•a + ℕℤ 2*b < ℕℤ 2*a ⇒ b + b < a⌝)
([], ⌜∀ x y:ℤ• ¬ (ℕℤ 2*x + y = ℕℤ 4 ∧ ℕℤ 4*x + ℕℤ 2*y = ℕℤ 7)⌝)
```

See Also *′ℤ_lin_arith*

SML
```
(* Proof Context: 'ℤ_lin_arith *)
```

Description This is a component proof context whose main purpose is to supply a decision procedure for problems in linear arithmetic for the HOL integers.

Contents The rewriting, theorem stripping and conclusion stripping components are as for the proof context *ℤ_lin_arith* but without any of the material from the proof context *predicates*. The automatic proof components are as for *lin_arith*. Other components are blank.

A typical use of the proof context would be to solve problems containing a mixture of (linear) arithmetic and set theory.

Errors
```
82200 Could not prove theorem with conclusion ?1 (?0)
```

SML
```
val ℤ_anf_conv : CONV;
val ℤ_ANF_C : CONV -> CONV;
```

Description *ℤ_anf_conv* is a conversion which proves theorems of the form ⊢ *t1* = *t2* where *t1* is a term formed from atoms of type ℤ and *t2* is in what we may call additive normal form, i.e. it has the form: $t_1 + t_2 + ...$, where the t_i have the form $s_1 * s_2 * ...$where the s_i are atoms. Here, by atom we mean a term which is not of the form $t_1 + t_2 + ...$or $s_1 * s_2 * ....$

The summands t_i and, within them, the factors s_j are given in increasing order with respect to the ordering on terms given by the function *ℤ_term_order*, q.v. Arithmetic computation is carried out on atoms to ensure that at most one of the summands is a numeric literal and that, within each summand, at most one factor is a numeric literal. Any literal appears at the beginning of its factor or summand and addition of *0* or multiplication by *1* is simplified out.

ℤ_ANF_C conv is a conversion which acts like *ℤ_anf_conv* but which applies *conv* to each atom as it is encountered (and normalises the result recursively). The argument conversion may signal that it does not wish to change a subterm, t say, either by failing or by returning $t = t$, the former approach is more efficient.

The conversions fail with error number 105032 if there are no changes to be made to the term.

Errors
```
105032 ?0 is not of type ⌜:ℤ⌝ or is already in additive normal form
```

SML
```
val ℤ_eq_cancel_conv : CONV
val ℤ_≤_cancel_conv : CONV
```

Description *ℤ_eq_cancel_conv* (resp. *ℤ_≤_cancel_conv*) puts arithmetic equations (resp. inequalities formed with ≤) in a normal form in which the right-hand side is a signed literal and the left-hand side is in additive normal form, in the sense of *ℤ_anf_conv*, q.v.

For example, the calls:

ProofPower Input
```
ℤ_eq_cancel_conv⌜x + ℕℤ 2*y + ℕℤ 3 = ℕℤ 1 + ℕℤ 6*y⌝;
ℤ_≤_cancel_conv⌜x + ℕℤ 2*y + x + ℕℤ  3 ≤ y + ℕℤ 2 + ℕℤ 2*x + y⌝;
```

produce the following output

ProofPower Output
```
val it = ⊢ x + ℕℤ 2 * y + ℕℤ 3 = ℕℤ 1 + ℕℤ 6 * y
        ⇔ x + ~ (ℕℤ 4) * y = ~ (ℕℤ 2) : THM
val it = ⊢ x + ℕℤ 2 * y + x + ℕℤ 3 ≤ y + ℕℤ 2 + ℕℤ 2 * x + y
        ⇔ ℕℤ 1 ≤ ℕℤ 0 : THM
```

Note that if the left-hand side reduces to *0* the truth value is not evaluated. However, *ℤ_eq_conv* or *ℤ_≤_conv* may be used to perform the evaluation, where required.

Uses The conversions are intended for use in tactic and conversion programming. The normal interactive interface is via rewriting or stripping in the proof context *ℤ_lin_arith*, which performs other useful simplifications.

Errors
```
105120 ?0 is not of the form (t1:ℤ) = t2 or is already in normal form
105121 ?0 is not of the form (t1:ℤ) ≤ t2 or is already in normal form
```

See Also *lin_arith*, *ℤ_lin_arith*, *ℤ_eq_conv*, *ℤ_≤_conv*

SML
```
val ℤ_lin_arith_rule : TERM list -> THM;
```

Description Given a system, $\Gamma = [r_1, r_2, ...]$, of numeric constraints, r_i, of the form $(s_i:\mathbb{Z}) = \mathbb{N}\mathbb{Z}\ 0$or $s_i \leq \mathbb{N}\mathbb{Z}\ 0$ these rules attempt to prove a theorem of the form $\Gamma \vdash F$. Terms in Γ which are not of either of these forms are ignored and do not appear in the assumptions of the result theorem.

The usual interface to these rules is via the decision procedures in the proof context *ℤ_lin_arith*, q.v., e.g. as invoked by *PC_T1"ℤ_lin_arith"prove_tac*[]

The algorithm for the decision procedure is very similar to the one used in *lin_arith_rule*, q.v. The only significant difference is that there is no opportunity with the integers to add in assumptions that all atoms are non-negative.

Errors
```
82110  System is satisfiable
82111  A system with no constraints is satisfiable
105112 No constraints of the form ⊢ (t1:ℤ) ≤ t2 or ⊢ (t1:ℤ) = t2 could
       be derived from the assumptions
```

SML
```
val ℤ_lin_arith_tac : TACTIC;
```

Description This tactic is primarily intended for use in conjunction with the integer linear arithmetic proof contexts *ℤ_lin_arith* and *'ℤ_lin_arith*, q.v. The normal interface to the tactics is via the decision procedures in these proof contexts, as in for example: *PC_T1* "*ℤ_lin_arith*" *prove_tac*[].

The tactics do however, have possible applications in specialised tactic programming, in which circumstances their behaviour may be understood from their definition, in terms of *ℤ_lin_arith_rule*, q.v., essentially as:

```
val ℤ_lin_arith_tac = GET_ASMS_T (f_thm_tac o ℤ_lin_arith_rule o map concl);
```

Uses The most likely application is in specialised tactic programming in situations where it is known that the assumptions are already in the normal form produced by the proof context *lin_arith* and it is important for performance not to restrip them.

SML
```
val ℤ_term_order : TERM -> TERM -> int;
```

Description *ℤ_term_order* gives an ordering relation on HOL terms analogous to that given by *term_order*, q.v., but which takes special arrangements for certain terms of type ℤ. In particular it ensures that a 'monomial' (formed using integer multiplication) without a sign immediately precedes the same monomial with a sign, which in turn precedes the same monomial with a signed literal multiplier, e.g.:

```
a*b < ~a * b < 0 * a * b < ~1 * a * b < 1 * a * b < ....
```

9.2.8 Lists

SML
```
signature List = sig
```

Description This is the signature in which the theory "list" is declared.

SML
```
val is_list_rep_def : THM
val nil_cons_def : THM
val length_def : THM
val hd_def : THM
val tl_def  : THM
val append_def : THM
val map_def : THM
val fold_def : THM
val split_def : THM
val combine_def : THM
val rev_def : THM
```

Description These Standard ML variables are bound to the definitions in the theory "list".

SML
```
val list_clauses : THM
val list_cases_thm : THM
val list_induction_thm : THM
val list_prim_rec_thm : THM
```

Description These Standard ML variables are bound to the theorems in the theory "list".

SML

val **list_induction_tac** : $TERM \;{-}{>}\; TACTIC$

Description This tactic implements induction over lists : to prove $t[x]$ it suffices to prove $t[Nil]$ and to prove $t[Cons\ h\ x]$ on the assumption that $t[x]$. The term argument must appear free in the conclusion of the goal but not in its assumptions.

Tactic

$$\frac{\{\ \Gamma\ \}\ t[x]}{\{\ \Gamma\ \}\ t[Nil]\ ;\ strip\{t[x],\ \Gamma\}\ t[Cons\ h\ x]} \qquad list\_induction\_tac\ \ulcorner x \urcorner$$

See Also *LIST_INDUCTION_T*

Errors

39001 ?0 is not a term variable
39002 ?0 does not appear free in the conclusion of the goal
39003 ?0 appears free in the assumptions of the goal

SML

val **LIST_INDUCTION_T** : $TERM \;{-}{>}\; (THM \;{-}{>}\; TACTIC) \;{-}{>}\; TACTIC;$

Description This implements induction over lists as a tactical. The term argument must appear free in the conclusion of the goal but not in its assumptions. The inductive hypothesis is passed to the tactic generating function given by the second argument.

Tactic

$$\frac{\{\ \Gamma\ \}\ t[x]}{\begin{array}{c}\{\ \Gamma\ \}\ t[Nil]\ ;\ ttac(t[x] \vdash t[x]) \\ (\{\ \Gamma\ \}\ t[Cons\ h\ x])\end{array}} \qquad LIST\_INDUCTION\_T\ \ulcorner x \urcorner\ ttac$$

Uses Most commonly used with *asm_tac* to avoid the stripping up of the inductive hypothesis which occurs with *list_induction_tac*.

See Also *list_induction_tac*

Errors

39001 ?0 is not a term variable
39002 ?0 does not appear free in the conclusion of the goal
39003 ?0 appears free in the assumptions of the goal

9.2.9 Characters and Strings

SML

signature **Char** = *sig*

Description This is the signature for the theory of characters and strings.

SML

val **is_char_rep_def** : *THM*
val **char_def** : *THM*
val **abs_char_rep_char_def** : *THM*

Description These Standard ML variables are bound to the definitions in the theory “char”.

9.2.10 Set Theory

SML

```
signature SetsTheory = sig
        val set_def : THM;
        val sets_clauses : THM;
        val ∅_clauses : THM;
        val ∈_in_clauses : THM;
        val sets_ext_clauses : THM;
        val complement_clauses : THM;
        val ∪_clauses : THM;
        val ∩_clauses : THM;
        val set_dif_clauses : THM;
        val ⊖_clauses : THM;
        val ⊆_clauses : THM;
        val ⊂_clauses : THM;
        val ⋃_clauses : THM;
        val ⋂_clauses : THM;
        val ℙ_clauses : THM;
        val is_set_rep_def : THM;
        val set_comp_def : THM;
        val insert_def : THM;
        val complement_def : THM;
        val ∪_def : THM;
        val ∩_def : THM;
        val set_dif_def : THM;
        val ⊖_def : THM;
        val ⊆_def : THM;
        val ⊂_def : THM;
        val ⋃_def : THM;
        val ⋂_def : THM;
        val ℙ_def : THM;
```

Description This is the signature in which we declare theory "sets". *sets_clauses* is bound the the theorem saved with the same name as key, *insert_def* is bound to the definition of *Insert*, saved with key *Insert*, and the other *_def* follow the same pattern.

SML

val **sets_eq_conv** : $CONV$;
val **⊆_conv** : $CONV$;
val **⊂_conv** : $CONV$;

Description These conversions use the extensional property of sets to transform terms of the form ⌜$a = b$⌝, ⌜$a \subseteq b$⌝ or ⌜$a \subset b$⌝ (where a and b are sets) into assertions about membership of a and b. The conversions introduce variable structures in the membership assertions if the elements of a and b are pairs (i.e. have types of the form: $\sigma \times \tau$).

If the elements are not pairs, the conversions behave as follows:

Conversion

$$\frac{}{\vdash (a = b) \Leftrightarrow (\forall\ x \bullet x \in a \Leftrightarrow x \in b)} \quad \begin{array}{l} sets\_eq\_conv \\ \ulcorner a = b \urcorner \end{array}$$

Conversion

$$\frac{}{\vdash (a \subseteq b) \Leftrightarrow (\forall\ x \bullet x \in a \Rightarrow x \in b)} \quad \begin{array}{l} \subseteq\_conv \\ \ulcorner a \subseteq b \urcorner \end{array}$$

Conversion

$$\frac{}{\begin{array}{r}\vdash (a \subset b) \Leftrightarrow (\forall\ x \bullet x \in a \Rightarrow x \in b) \\ \land (\exists\ x \bullet \neg\ x \in a \land x \in b)\end{array}} \quad \begin{array}{l} \subset\_conv \\ \ulcorner a \subset b \urcorner \end{array}$$

where x represents a variable whose name is "x" decorated to avoid clashing with the free variables of a or b.

If the elements of a and b are pairs, then a variable structure (whose free variables have names chosen from the sequence "$x1$", "$x2$", ... so as not to clash with any of the variables in a or b) is used instead of x on the left hand side of each $\in$, and the quantification is over the free variables of the variable structure. For example:

Example

$$\frac{}{\begin{array}{c}\vdash \{((1,\ 2),\ 3)\} \subseteq x2 \\ \Leftrightarrow \forall\ x1\ x3\ x4 \bullet \\ ((x1,\ x3),\ x4) \in \{((1,\ 2),\ 3)\} \\ \Rightarrow ((x1,\ x3),\ x4) \in x2\end{array}} \quad \begin{array}{l} \subseteq\_conv \\ \ulcorner \{(1,2),3\} \subseteq x2 \urcorner \end{array}$$

Here, $x2$ has not been used in the variable structure since it occurs in the input term.

(The distinction between the variable naming rules followed in the simple case and the paired case has some mnemonic value and makes the conversions compatible with rewriting with the theorem *sets_ext_clauses* as done, for example, in the proof context *sets_ext*.)

Errors

44021 ?0 is not of the form: ⌜$a = b$⌝ *where* ⌜a⌝ *and* ⌜b⌝ *are sets*
44022 ?0 is not of the form: ⌜$a \subseteq b$⌝
44023 ?0 is not of the form: ⌜$a \subset b$⌝

SML

```
val sets_simple_∃_conv : CONV;
```

Description This conversion changes an existentially quantified term in a (perhaps nullary) function whose range is a set to an existentially quantified term in a function whose range is a function from the type of the set element to ⌜ : $BOOL$⌝. It is only applicable if all instances of the bound function in the body of the term are applied to sufficent arguments and then tested for set membership by ∈.

Conversion

$$\frac{}{\vdash \exists f : ... \to ty\ SET \bullet P[\ t1 \in f,\ t2 \in f\ ...,\ ...] \Leftrightarrow \exists f' : ... \to ty\ SET \bullet P[\ f'\ ...\ t1,\ f'\ ...\ t2,\ ...]^{\urcorner}}$$

sets_simple_∃_conv
⌜$\exists f : ... \to ty\ SET \bullet P[\ t1 \in f\ ...,\ t2 \in f\ ...\ ,\ ...]$⌝

where the t_i are arbitrary terms.

Example

```
sets_simple_∃_conv ⌜∃ S • ∀ x y • x ∈ S y ⇔ ¬(x = y)⌝ --->
 ⊢ (∃ S • ∀ x y • x ∈ S y ⇔ ¬(x = y)) ⇔
   (∃ S′ • ∀ x y • S′ y x ⇔ ¬(x = y))
```

Uses This function is used to implement a automated existence proof preprocessor for the "sets" proof context.

Errors

44010 ?0 is not of the form: ⌜∃ x : ... → ty SET • ...⌝
44011 ?0 is not of the form: ⌜∃ x • ... x ...⌝
44012 ?0 has instances of ?1 not embedded in subterms of the form: ⌜t ∈ ?1 ...⌝
44013 Unable to prove ?0

SML

```
val simple_∈_comp_conv : CONV;
```

Description A conversion for set membership. It cannot handle variable structures bound by the set comprehension.

Conversion

$$\frac{}{\vdash x \in \{v \mid p[v]\} \Leftrightarrow p[x]}$$

simple_∈_comp_conv
⌜$x \in \{v \mid p[v]\}$⌝

See Also *∈_comp_conv*

Errors

44001 ?0 is not of the form:⌜x ∈ {v | p[v]}⌝

SML
```
val ∈_comp_conv : CONV;
```

Description A conversion for set membership. It can handle variable structures bound by the set comprehension.

Conversion

$$\frac{}{\vdash t \in \{vs[x,y,...] \mid p[x,y,...]\} \Leftrightarrow p[x1,y1,...]} \qquad \begin{array}{l} \in\_comp\_conv \\ \ulcorner t \in \{v[x,y,...] \mid p[x,y,...]\}\urcorner \end{array}$$

where $x1$, $y1$, etc, are the appropriate components of t, extracted via *Fst* and *Snd*.

See Also *simple_∈_comp_conv*

Errors
```
27002 ?0 is not of form:⌜x ∈ {v | p[v]}⌝
```

SML
```
val ∈_enum_set_conv : CONV;
```

Description Give that something within an enumerated set is a member of that set.

Conversion

$$\frac{}{\vdash ti \in \{t1,...,tn\} \Leftrightarrow T} \qquad \begin{array}{l} \in\_enum\_set\_conv \\ \ulcorner ti \in \{t1,...,tn\}\urcorner \end{array}$$

where ti is α-convertible to one of $t1, \ldots, tn$.

Errors
```
27005 ?0 not a member of the enumerated set ?1
27006 ?0 is not of the form: ⌜x ∈ {t1, ..., tn}⌝
```

Message 3012 occurs when the term list cannot be made into an enumerated set because of differing types.

SML
```
val ∈_enum_set_rule : TERM -> TERM list -> THM;
```

Description Give that something within an enumerated set is a member of that set.

Rule

$$\frac{}{\vdash ti \in \{t1,...,tn\}} \qquad \begin{array}{l} \in\_enum\_set\_rule \\ \ulcorner ti\urcorner \\ [\ulcorner t1\urcorner,...\ulcorner tn\urcorner] \end{array}$$

where ti is α-convertible to one of $t1, \ldots, tn$.

Errors
```
3012  ?0 and ?1 do not have the same types
27001 ?0 not a member of list of terms
```

Message 3012 occurs when the term list cannot be made into an enumerated set because of differing types.

9.2.11 One, Combin, and Sum

SML
```
signature BasicHolTheory = sig
end;
```

Description This is the signature in which we declare theory "basic_hol". The theory itself is empty.

SML
```
signature CombinTheory = sig
        val comb_s_def : THM;
        val comb_k_def : THM;
        val comb_i_def : THM;
        val o_def : THM;
        val o_assoc_thm : THM;
        val o_i_thm : THM;
end;
```

Description In the signature of *CombinTheory* in which we declare theory "combin" we bind the definitions of *CombS*, *CombK*, *CombI* and *o* to *comb_s_def*, *comb_k_def*, *comb_i_def* and *o_def* respectively. The theorem saved with key "*o_assoc_thm*" is bound to the ML value of the same name, as is "*o_i_thm*".

SML
```
signature HolTheory = sig
end;
```

Description This is the signature in which we declare theory "hol". The theory itself is empty.

SML
```
signature OneTheory = sig
        val is_one_rep_def : THM;
        val one_def : THM;
        val one_def1 : THM;
        val one_fns_thm : THM;
end;
```

Description This is the signature in which we declare theory "one". *one_def* is the definition of *One*, saved with the key "One",*one_def1* is the type definition of *ONE*, saved with the key "ONE". . *one_fns_thm* is the theorem saved with key "*one_fns_thm*".

SML
```
signature SumTheory = sig
        val is_sum_rep_def : THM;
        val sum_def : THM;
        val sum_clauses : THM;
        val sum_cases_thm : THM;
        val sum_fns_thm : THM;
```

Description *sum_clauses* is bound to the definition of *InL*. *sum_cases_thm* and *sum_fns_thm* are bound to the theorems with those names as their key.

SML

val **sum_cases_tac** : *TERM* −> *TACTIC*;

Description This tactic implements case splitting over sum types: to prove $t[x]$ it suffices to prove $t[InL y]$ and to prove $t[InR z]$ (for newly introduced variable y and z). The term argument must be a variable of type ⌜:ty1 + ty2⌝ and must appear free in the conclusion of the goal but not in its assumptions.

Tactic

$$\frac{\Gamma \vdash P\ [x]}{\Gamma \vdash P\ [InL\ x'],\quad \Gamma \vdash P\ [InR\ x'']} \qquad \begin{array}{l} sum\_cases\_tac \\ \ulcorner x : ty1 + ty2 \urcorner \end{array}$$

where x' and x'' are variables whose names are variants of x, and types are *ty1* and *ty2* respectively.

Errors

45002 ?0 is not a variable of type ⌜:ty1 + ty2⌝
38002 ?0 does not appear free in the conclusion of the goal
38003 ?0 appears free in the assumptions of the goal

SML

val ∀_**sum_conv** : *CONV*;

Description A conversion to break a term that is universally quantified by a sum type variable into two terms universally quantified by variables of the constituent types.

Conversion

$$\frac{}{\vdash (\forall\ x : ty1 + ty2 \bullet P[x]) \Leftrightarrow (\forall\ x' : ty1 \bullet P[InL\ x']) \land (\forall\ x'' : ty2 \bullet P[InR\ x''])} \qquad \begin{array}{l} \forall\_sum\_conv \\ \ulcorner \forall\ x : ty1 + ty2 \bullet P[x] \urcorner \end{array}$$

where x' and x'' are variables whose names are variants of x, and types are *ty1* and *ty2* respectively.

Errors

45001 ?0 is not of the form ⌜∀ x : ty1 + ty2 • P[x]⌝

9.2.12 Binary Relations

SML

signature **BinRelTheory** = *sig*
val $\mapsto$**_def** : *THM*;
val $\times$**_def** : *THM*;
val $\leftrightarrow$**_def** : *THM*;
val **dom_def** : *THM*;
val **ran_def** : *THM*;
val **id_def** : *THM*;
val $⨟$**_def** : *THM*;
val **r_**$⨟$**_r_def** : *THM*;
val **r_o_r_def** : *THM*;
val $\lhd$**_def** : *THM*;
val $\rhd$**_def** : *THM*;
val $⩤$**_def** : *THM*;
val $⩥$**_def** : *THM*;
val **graph_def** : *THM*;
val **inv_rel_def** : *THM*;
val **image_def** : *THM*;
val $\oplus$**_def** : *THM*;
val **reflexive_def** : *THM*;
val **symmetric_def** : *THM*;
val **transitive_def** : *THM*;
val **injective_def** : *THM*;
val **surjective_def** : *THM*;
val **total_def** : *THM*;
val **functional_def** : *THM*;
val **tc_def** : *THM*;
val **rtc_def** : *THM*;
val **rel_combine_def** : *THM*;
val **rel_**$\in$**_in_clauses** : *THM*;
val **bin_rel_ext_clauses** : *THM*;
val **inv_rel_thm** : *THM*;
val **bin_rel_**$\varnothing$**_universe_thm** : *THM*;
val **bin_rel_insert_thm** : *THM*;

Description This is the signature in which we declare theory "*bin_rel*".

SML

(* *Proof context key* "'*bin_rel*" *)

Description A component proof context for theory *bin_rel*, "aggressively" converting problems involving the vocabulary of the theory *bin_rel* into that of the theory *sets*.

Contents Rewriting: *rel_∈_in_clauses*, *inv_rel_thm* and the definitions of the constants defined in the theory *bin_rel*. Stripping theorems: *rel_∈_in_clauses*, *inv_rel_thm*(also pushed in through '¬').

Stripping conclusions: *rel_∈_in_clauses*, *inv_rel_thm*(also pushed in through '¬').

Automatic proof procedures are *basic_prove_tac* and *basic_prove_conv*.

Usage Notes Should not be used in conjunction with component proof context "'*sets_ext*", or with a complete proof context which includes it. Requires theories *bin_rel* and *sets*. Intended for use in conjunction with a proof context such as *sets_alg* or *sets_ext* which is capable of solving the resulting set-theoretic conjectures.

9.2.13 Functional Relations

SML

```
signature FunRelTheory = sig
        val ⇸_def : THM;
        val →_def : THM;
        val ⤔_def : THM;
        val ↣_def : THM;
        val ⤀_def : THM;
        val ↠_def : THM;
        val ⤖_def : THM;
        val at_def : THM;
        val graph_at_thm : THM;
        val inv_rel_∈_arrow_thm : THM;
        val at_at_eq_thm : THM;
```

Description This is the signature in which we declare theory "fun_rel".

SML

(* *proof context key* "'*fun_rel_ext*" *)

Description This proof context extends '*bin_rel_ext* by:

Name

Becomes "fun_rel_ext".

Stripping Goals and Theorems

Adding in ? applied at the top level or under a single negation.

Rewriting Context

Adding in ?.

SML
```
(∗ proof context key "fun_rel_alg" ∗)
```

Description *fun_rel_alg* extends *relation_alg* by:

Name

Becomes “fun_rel_alg”.

Stripping Goals and Theorems

Adding in ? applied at the top level or under a single negation.

Rewriting Context

Adding in ?.

9.2.14 Finite Sets

SML
```
signature FinSetTheory = sig
	val ℕ_def : THM;	val finite_def : THM;
	val 𝔽_def : THM;	val 𝔽1_def : THM;
	val min_def : THM;	val max_def : THM;
	val size_def : THM;	val iter_def : THM;
	val ⇻_def : THM;	val ⤀_def : THM;
	val finite_induction_thm : THM;	val empty_finite_thm : THM;
	val singleton_∪_finite_thm : THM;	val ⊆_finite_thm : THM;
	val ∪_finite_thm : THM;	val ∩_finite_thm : THM;
	val finite_distinct_elems_thm : THM;	val length_↾_≤_thm : THM;
	val length_↾_less_thm : THM;	val elems_↾_thm : THM;
	val distinct_length_≤_thm : THM;	val distinct_size_length_thm : THM;
	val size_empty_thm : THM;	val size_singleton_∪_thm : THM;
	val size_singleton_thm : THM;	val size_∪_thm : THM;
	val size_0_thm : THM;	val size_1_thm : THM;
	val ⋃_finite_thm : THM;	val pigeon_hole_thm : THM;
	val ⊆_size_≤_thm : THM;	val ⊆_size_less_thm : THM;
	val min_∈_thm : THM;	val min_≤_thm : THM;
	val max_∈_thm : THM;	val ≤_max_thm : THM;
	val finite_⊆_well_founded_thm : THM
```

Description These are the ML bindings for the definitions and theorems in the declare theory “*fin_set*”.

SML
```
val finite_induction_tac : TERM -> TACTIC;
```

Description An induction tactic for finite sets. To prove t on the assumption that $s \in Finite$, it suffices to prove $t[\{\}/s]$ and to prove $t[(\{x\} \cup s)/s]$ on the assumption that t holds, that $s \in Finite$, and that $\neg x \in s$. The term argument must be a variable, s, with type an instance of ⌜:'a SET⌝ and must appear free in the conclusion of the goal. It must also appear once, and only once, in an assumption of the form $s \in Finite$.

Tactic

$$\frac{\{\ \Gamma,\ s \in Finite\}\ t[s]}{\{\ \Gamma\ \}\ t[\{\}/s]\ ;\quad strip\ \{t,\ s \in Finite,\ \neg x \in s,\ \Gamma\}\ t[(\{x\} \cup s)/s]} \qquad finite\_induction\_tac\ ⓩs⌝$$

Errors

73001	*?0 is not a variable*
73002	*A term of the form ⓩv ∈ Finite⌝ where v is the induction variable could not be found in the assumptions*
73003	*?0 does not appear free in the conclusion of the goal*
73004	*?0 appears free in more than one assumption of the goal*

9.2.15 Sequences

SML
```
signature SeqTheory = sig
	val elems_def : THM;
	val distinct_def : THM;
	val lists_def : THM;
	val lists1_def : THM;
	val inj_lists_def : THM;
	val nth_def : THM;
	val dot_dot_def : THM;
	val list_rel_def : THM;
	val rel_list_def : THM;
	val ⁀_def : THM;
	val head_def : THM;
	val last_def : THM;
	val front_def : THM;
	val tail_def : THM;
	val ↾_def : THM;
	val enumerate_def : THM;
	val squash_def : THM;
	val extract_def : THM;
	val prefix_def : THM;
	val suffix_def : THM;
	val in_def : THM;
	val flat_def : THM;
```

Description This is the signature in which we declare theory "funrel".

SML

```
(* proof context key "seq_ext" *)
```

Description *seq_ext* extends *finset_ext* by:

Name

Becomes “finset_ext”.

Stripping Goals and Theorems

Adding in ? applied at the top level or under a single negation.

Rewriting Context

Adding in ?.

SML

```
(* proof context key "funrel_alg" *)
```

Description *finset_alg* extends *funrel_alg* by:

Name

Becomes “finset_alg”.

Stripping Goals and Theorems

Adding in ? applied at the top level or under a single negation.

Rewriting Context

Adding in ?.

9.2.16 Real Numbers

SML

(∗ *Proof Context*: $'\mathbb{R}$ ∗)

Description A component proof context for handling the basic arithmetic operations for real numbers in HOL.

Expressions and predicates treated by this proof context are constructs formed from:

$+$, $*$, $-$, Abs_R, $/_R$, $\leq$, $<$, $\geq$, $>$, $=$

Contents

Rewriting:

$\mathbb{R}$_*plus_conv*, $\mathbb{R}$_*times_conv*, $\mathbb{R}$_*subtract_minus_conv*
$\mathbb{R}$_*abs_conv*, $\mathbb{R}$_*over_conv*, $\mathbb{R}$_*recip_conv*, $\mathbb{R}$_$\mathbb{N}$_*exp_conv*, $\mathbb{R}$_$\mathbb{Z}$_*exp_conv*,
$\mathbb{R}$_*eq_conv*, $\mathbb{R}$_≤_*conv*, $\mathbb{R}$_*less_conv*
$\mathbb{R}$_≥_*conv*, $\mathbb{R}$_*greater_conv*,
$\mathbb{R}$_*plus_clauses*, $\mathbb{R}$_*minus_clauses*, $\mathbb{R}$_≤_*clauses*
$\mathbb{R}$_*less_clauses*, $\mathbb{R}$_*frac_norm_conv*, *float_conv*

Here *float_conv* is only applied to floating point literals with a zero exponent.

Stripping theorems:

$\mathbb{R}$_*eq_conv*, $\mathbb{R}$_≤_*conv*, $\mathbb{R}$_*less_conv*
$\mathbb{R}$_≥_*conv*, $\mathbb{R}$_*greater_conv*,
$\mathbb{R}$_*plus_clauses*, $\mathbb{R}$_*minus_clauses*, $\mathbb{R}$_≤_*clauses*
$\mathbb{R}$_*less_clauses*,
and all the above pushed through ¬

Stripping conclusions: as for stripping theorems.

Rewriting canonicalisation: blank.

Automatic proof procedures: *basic_prove_tac*, *basic_prove_conv*.

Automatic existence prover: blank.

SML

(∗ *Proof Context*: ℝ_**lin_arith** ∗)

Description This is a proof context whose main purpose is to supply a decision procedure for problems in linear arithmetic over the HOL integers.

Contents The rewriting, theorem stripping and conclusion stripping components are as for the proof context *predicates*, q.v, each augmented with conversions effecting the following transformations, where a and b stand for numeric literals.

$$
\begin{array}{lll}
a = b & \to & ℝ\_eq\_conv⌜a = b⌝ \\
a \le b & \to & ℝ\_\le\_conv⌜a \le b⌝ \\
t1 = t2 & \to & ℝ\_eq\_cancel\_conv⌜t1 = t2⌝ \\
t1 \le t2 & \to & ℝ\_\le\_cancel\_conv⌜t1 \le t2⌝ \\
\neg\ t1 = t2 & \to & t1 < t2 \lor t2 < t1 \\
\neg\ t1 \le t2 & \to & t2 < t1 \\
t1 < t2 & \to & t1 + 1 \le t2 \\
t1 \ge t2 & \to & t2 \le t1 \\
t1 > t2 & \to & t2 < t \\
\neg\ t1 < t2 & \to & t2 \le t1 \\
\neg\ t1 \ge t2 & \to & \neg\ t2 \le t1 \\
\neg\ t1 > t2 & \to & t1 \le t2 \\
t1 = t2 & \to & t1 = t2 \\
t1 \le t2 & \to & t1 \le t2
\end{array}
$$

(where all variables are of type ℝ).

The automatic proof tactic works by *(i)* restripping all the assumptions of the goal, *(ii)* adding the argument theorems to the stock of assumptions using *strip_asm_tac*, *(iii)* applying *contr_tac*, and *(iv)* searching for a linear combination of the assumptions which will reduce, by multiplying out and cancelling like terms, to a contradiction of the form $a = b$ or $a \le b$ with a and b numeric literals. The automatic proof conversion just tries to prove its argument, t say, using the automatic proof tactic and returns $t = T$ if it succeeds.

Other components of the proof context are as for *predicates*.

For example, *PC_T1*"ℝ_*lin_arith*"*prove_tac*[] will prove any of the following goals:

$$
\begin{array}{l}
([], ⌜\forall a\ b\ c{:}ℝ\bullet a \le b \land (a + b < c + a) \Rightarrow a < c⌝) \\
([], ⌜\forall a\ b\ c{:}ℝ\bullet a \ge b \land \neg\ b < c \Rightarrow (1/2) * a \ge (1/2) * c⌝) \\
([], ⌜\forall a\ b\ c{:}ℝ\bullet a + 2.{*}b < 2.{*}a \Rightarrow b + b < a⌝) \\
([], ⌜\forall\ x\ y{:}ℝ\bullet \neg\ (2.{*}x + y = 1/3 \land 4.{*}x + 2.{*}y = 2/5)⌝)
\end{array}
$$

See Also 'ℝ_*lin_arith*

SML

(∗ *Proof Context*: 'ℝ_**lin_arith** ∗)

Description This is a component proof context whose main purpose is to supply a decision procedure for problems in linear arithmetic for the HOL integers.

Contents The rewriting, theorem stripping and conclusion stripping components are as for the proof context ℝ_*lin_arith* but without any of the material from the proof context *predicates*. The automatic proof components are as for *lin_arith*. Other components are blank.

A typical use of the proof context would be to solve problems containing a mixture of linear real number arithmetic and set theory.

SML

```
val dest_ℝ_less : TERM -> TERM * TERM;   (* <_R *)
val dest_ℝ_≤ : TERM -> TERM * TERM             (* ≤_R *);
val dest_ℝ_greater : TERM -> TERM * TERM        (* >_R *);
val dest_ℝ_≥ : TERM -> TERM * TERM             (* ≥_R *);
val dest_ℝ_plus : TERM -> TERM * TERM   (* +_R *);
val dest_ℝ_subtract : TERM -> TERM * TERM      (* -_R *);
val dest_ℝ_times : TERM -> TERM * TERM  (* *_R *);
val dest_ℝ_over : TERM -> TERM * TERM   (* /_R *);
val dest_ℝ_frac : TERM -> TERM * TERM   (* /_N *);
val dest_ℝ_minus : TERM -> TERM         (* ~_R *);
val dest_ℝ_abs : TERM -> TERM                  (* Abs_R *);
val dest_ℝ_recip : TERM -> TERM                (* ^-1 *);
val dest_ℝ_ℕ_exp : TERM -> TERM * TERM;(* ^_N *)
val dest_ℝ_ℤ_exp : TERM -> TERM * TERM;(* ^_Z *)
val dest_ℕℝ : TERM -> TERM;          (* ℕℝ *)
val dest_ℝ_max : TERM -> TERM                  (* Max_R *);
val dest_ℝ_min : TERM -> TERM                  (* Min_R *);
```

Description These are the derived destructor functions for the basic arithmetic operators on the real numbers.

Errors

116101 ?0 is not of the form $\ulcorner x <_R y \urcorner$
116102 ?0 is not of the form $\ulcorner x \leq_R y \urcorner$
116103 ?0 is not of the form $\ulcorner x >_R y \urcorner$
116104 ?0 is not of the form $\ulcorner x \leq_R y \urcorner$
116105 ?0 is not of the form $\ulcorner x +_R y \urcorner$
116106 ?0 is not of the form $\ulcorner x -_R y \urcorner$
116107 ?0 is not of the form $\ulcorner x *_R y \urcorner$
116108 ?0 is not of the form $\ulcorner x /_R y \urcorner$
116109 ?0 is not of the form $\ulcorner x /_N y \urcorner$
116111 ?0 is not of the form $\ulcorner \sim_R y \urcorner$
116112 ?0 is not of the form $\ulcorner Abs_R\ x \urcorner$
116113 ?0 is not of the form $\ulcorner x\ ^{-1} \urcorner$
116114 ?0 is not of the form $\ulcorner x\ \hat{}_N\ m \urcorner$
116115 ?0 is not of the form $\ulcorner \mathbb{NR}\ m \urcorner$
116116 ?0 is not of the form $\ulcorner x\ \hat{}_Z\ i \urcorner$
116117 ?0 is not of the form $\ulcorner Max_R\ L \urcorner$
116118 ?0 is not of the form $\ulcorner Min_R\ L \urcorner$

SML
```
val float_conv : CONV;
```

Description These conversions compute theorems of the form and

$\vdash Max_R\ x = y$
$\vdash Min_R\ x = y$

where x is either a singleton list display or a list display or list-valued expression formed with *Cons* whose first two elements are rational literals. E.g.,

$\vdash Max_R\ [4\ /\ 3;\ 1.0;\ 2\ /\ 3] = Max_R\ [4\ /\ 3;\ 2\ /\ 3]$
$-TEX$

Errors
116331 Could not simplify ?0

SML
```
val is_ℝ_less : TERM -> bool          (* <_R *);
val is_ℝ_≤ : TERM -> bool             (* ≤_R *);
val is_ℝ_greater : TERM -> bool (* >_R *);
val is_ℝ_≥ : TERM -> bool             (* ≥_R *);
val is_ℝ_plus : TERM -> bool          (* +_R *);
val is_ℝ_subtract : TERM -> bool      (* −_R *);
val is_ℝ_times : TERM -> bool         (* *_R *);
val is_ℝ_over : TERM -> bool          (* /_R *);
val is_ℝ_frac : TERM -> bool      (* /_N *);
val is_ℝ_minus : TERM -> bool     (* ∼_R *);
val is_ℝ_abs : TERM -> bool           (* Abs_R *);
val is_ℝ_recip : TERM -> bool     (* ^-1 *);
val is_ℝ_ℕ_exp : TERM -> bool     (* ^_N *);
val is_ℝ_ℤ_exp : TERM -> bool     (* ^_Z *);
val is_ℕℝ : TERM -> bool          (* ℕℝ *);
val is_ℝ_max : TERM -> bool           (* Max_R *);
val is_ℝ_min : TERM -> bool           (* Min_R *);
```

Description These are the derived discriminator functions for the basic arithmetic operators on the real numbers. The ML comments above give the name of the corresponding operators. These names all have aliases without the subscript.

SML
```
val mk_ℝ_less : TERM * TERM -> TERM;     (* <_R *)
val mk_ℝ_≤ : TERM * TERM -> TERM         (* ≤_R *);
val mk_ℝ_greater : TERM * TERM -> TERM (* >_R *);
val mk_ℝ_≥ : TERM * TERM -> TERM         (* ≥_R *);
val mk_ℝ_plus : TERM * TERM -> TERM      (* +_R *);
val mk_ℝ_subtract : TERM * TERM -> TERM  (* −_R *);
val mk_ℝ_times : TERM * TERM -> TERM     (* *_R *);
val mk_ℝ_over : TERM * TERM -> TERM      (* /_R *);
val mk_ℝ_frac : TERM * TERM -> TERM      (* /_N *);
val mk_ℝ_minus : TERM -> TERM            (* ∼_R *);
val mk_ℝ_abs : TERM -> TERM              (* Abs_R *);
val mk_ℝ_recip : TERM -> TERM            (* ^-1 *);
val mk_ℝ_ℕ_exp : TERM * TERM -> TERM; (* ^_N *)
val mk_ℝ_ℤ_exp : TERM * TERM -> TERM; (* ^_Z *)
val mk_ℕℝ : TERM -> TERM;                (* ℕℝ*)
val mk_ℝ_max : TERM -> TERM              (* Max_R *);
val mk_ℝ_min : TERM -> TERM              (* Min_R *);
```

Description These are the derived constructor functions for the basic arithmetic operators on the real numbers. The ML comments above give the name of the corresponding operators. These names all have aliases without the subscript.

Both operands of *mk_ℝ_frac*and the second operand of *mk_ℝ_ℕ_exp* must be of type ℕ. All the other operands of type ℝ. All other operands must have type ℝ.

Errors
```
116201 ?0 does not have type ⌜:ℝ⌝
116202 ?0 does not have type ⌜:ℕ⌝
116203 ?0 does not have type ⌜:ℝ LIST⌝
```

SML
```
val ℝ_abs_conv : CONV;    val ℝ_recip_conv : CONV;
```

Description These conversions compute theorems of the form:

```
⊢ Abs x = z
⊢ x ^-1 = z
```

where x and z are rational literals (seeℝ_*plus_conv*.

Errors
```
116307 cannot take the reciprocal of ?0 because its denominator
       or numerator is 0
116308 ?0 is not of the form ⌜x ^-1⌝ where ⌜x⌝ is
       a rational literal
116316 ?0 is not of the form ⌜Abs (i/m)⌝
       where ⌜m⌝ is a natural number literal
```

SML

```
val ℝ_ANF_C : CONV -> CONV;
val ℝ_anf_conv : CONV;
```

Description ℝ_*anf_conv* is a conversion which proves theorems of the form ⊢ *t1* = *t2* where *t1* is a term formed from atoms of type ℝ and *t2* is in what we may call additive normal form, i.e. it has the form: $t_1 + t_2 + ...$, where the t_i have the form $s_1 * s_2 * ...$where the s_i are atoms. Here, by atom we mean a term which is not of the form $t_1 + t_2 + ...$or $s_1 * s_2 * ....$

The summands t_i and, within them, the factors s_j are given in increasing order with respect to the ordering on terms given by the function ℝ_*term_order*, q.v. Arithmetic computation is carried out on atoms to ensure that at most one of the summands is a rational literal and that, within each summand, at most one factor is a rational literal. Any literal appears at the beginning of its factor or summand and addition of *0* or multiplication by *1* is simplified out. Floating point literals are converted into rational literals.

ℝ_*ANF_C conv* is a conversion which acts like ℝ_*anf_conv* but which applies *conv* to each atom as it is encountered (and normalises the result recursively). The argument conversion may signal that it does not wish to change a subterm, *t* say, either by failing or by returning $t = t$, the former approach is more efficient.

The conversions fail with error number 116318 if there are no changes to be made to the term.

Errors

116318 ?0 is not of type ⌜:ℝ⌝ or is already in additive normal form

SML

```
val ℝ_copah_id_thm : THM;           val ℝ_copah_double_thm : THM;
val ℝ_copah_halve_thm : THM;        val ℝ_copah_comp_thm : THM;
val ℝ_copah_sum_thm : THM;          val ℝ_halve_closed_dense_thm : THM;
val ℝ_copah_dense_thm : THM;
```

Description An order-preserving additive homomorphism from ℝ to ℝ is said to be “central” if it commutes with any homomorphism from ℝ to ℝ. These are the ML bindings for theorems giving various properties of central order-preserving additive from ℝ to ℝ.

SML

```
val ℝ_delta_induction_thm : THM;
```

Description This is the theorem that supports ℝ_*delta_induction_tac* (q.v.).

SML

```
val ℝ_delta_induction_tac : TERM -> TACTIC;
```

Description This is an induction-like tactic that can be used to prove that a property holds in a half-open interval of the real numbers: to prove that $t[y]$holds whenever $x < y$, it suffices to exhibit a positive number d suc that *(i)* $t[u/y]$ holds for each u with $x < u < x + e$ where $d < e$ and *(ii)* $t[s + d/y]$ holds if $t[s/y]$ holds, for any s with $x < s$.

The term argument y must be a variable that appears free in the conclusion of the goal. The variable must also appear free once, and only once, in the assumptions, in an assumption of the form $x < y$.

Tactic

$$\frac{\{\ \Gamma;\ x < y\ \}\ t\ [y/y]}{\begin{array}{c}\{\ \Gamma\ \}\ \exists d \bullet 0. < d \\ \land\ (\exists e \bullet d < e\ \land \\ (\forall u \bullet x < u \land u < x + e \Rightarrow \\ t[u/y])) \\ \land\ (\forall s \bullet x < s \land p\ s \Rightarrow \\ t[s + d/y]))\end{array}} \qquad \mathbb{R}\_delta\_induction\_tac\ \ulcorner y \urcorner$$

Errors

116001 *?0 is not a variable of type* ⌜:ℝ⌝
116002 *A term of the form* ⌜$_Z$ *x < y*⌝ *where y is the induction variable could not be found in the assumptions*
116003 *?0 does not appear free in the conclusions of the goal*
116004 *?0 appears free in more than one assumption of the goal*

SML
```
val ℝ_eq_cancel_conv : CONV
val ℝ_≤_cancel_conv : CONV
val ℝ_less_cancel_conv : CONV
```

Description *ℝ_eq_cancel_conv* (resp. *ℝ_≤_cancel_conv*, resp. *ℝ_less_cancel_conv*) puts arithmetic equations (resp. inequalities formed with ≤) in a normal form in which the right-hand side is a signed literal and the left-hand side is in additive normal form, in the sense of *ℝ_anf_conv*, q.v.

For example, the calls:

ProofPower Input
```
ℝ_eq_cancel_conv⌜x + 2. * y + 3. = 1. + 6.*y⌝;
ℝ_≤_cancel_conv⌜x + 2. * y + x + . 3 ≤ y + 2. + 2.*x + y⌝;
ℝ_less_cancel_conv⌜x + 2. * y + x + . 3 < y + 2. + 2.*x + y⌝;
```

produce the following output

ProofPower Output
```
val it = ⊢ x + 2. * y + 3. = 1. + 6. * y
        ⇔ x + ~ (4.) * y = ~ (2.) : THM
val it = ⊢ x + 2. * y + x + 3. ≤ y + 2. + 2. * x + y
        ⇔ 1. ≤ 0. : THM
val it = ⊢ x + 2. * y + x + 3. ≤ y + 2. + 2. * x + y
        ⇔ 1. < 0. : THM
```

Note that if the left-hand side reduces to *0* the truth value is not evaluated. However, *ℝ_eq_conv*, *ℝ_≤_conv* or *ℝ_less_conv* may be used to perform the evaluation, where required.

Uses The conversions are intended for use in tactic and conversion programming. The normal interactive interface is via rewriting or stripping in the proof context *ℝ_lin_arith*, which performs other useful simplifications.

Errors
```
116520 ?0 is not of the form (t1:ℝ) = t2 or is already in normal form
116521 ?0 is not of the form (t1:ℝ) ≤ t2 or is already in normal form
116522 ?0 is not of the form (t1:ℝ) < t2 or is already in normal form
```

See Also *ℝ_lin_arith* (*proof context*)

SML
```
val ℝ_eval_conv : CONV;   val ℝ_EVAL_C : CONV -> CONV;
```

Description *ℝ_eval_conv* computes theorems of the form ⊢ *t1* = *t2* where *t1* is an expression made up from rational literals (see *ℝ_plus_conv*) and/or floating point literals using real addition, subtraction, multiplication, division, reciprocal, absolute value and unary negation. *t2* will be an optionally signed rational literal in normal form. The conversion fails if the expression cannot be evaluated (e.g., because it contains variables).

ℝ_EVAL_C conv is similar to *ℝ_eval_conv* but it also applies *conv* to any subterm that cannot be evaluated using the conversions for the arithmetic operations listed above. E.g., *ℝ_EVAL_C ℝ_ℕ_exp_conv* will evaluate expressions involving the usual arithmetic operations and also exponentiation of rational literals by natural number literals.

Errors
```
116320 ?0 cannot be evaluated
```

SML

```
val ℝ_frac_cross_mult_eq_thm : THM;
val ℝ_frac_cancel_thm : THM;
val ℝ_frac_0_thm : THM;
val ℝ_frac_ℕ_thm : THM;
val ℝ_frac_plus_frac_thm : THM;
val ℝ_frac_minus_frac_thm : THM;
val ℝ_frac_minus_frac_thm1 : THM;
val ℝ_over_times_over_thm : THM;
val ℝ_frac_times_frac_thm : THM;
val ℝ_over_recip_thm : THM;
val ℝ_frac_recip_thm : THM;
val ℝ_minus_recip_thm : THM;
val ℝ_over_eq_0_thm : THM;
val ℝ_over_over_over_thm : THM;
val ℝ_frac_less_frac_thm : THM;
val ℝ_minus_frac_less_frac_thm : THM;
val ℝ_frac_less_minus_frac_thm : THM;
val ℝ_0_≤_frac_thm : THM;
val ℝ_abs_frac_thm : THM;
val ℝ_abs_minus_thm : THM;
```

Description These are the ML bindings for some theorems about rational numbers (in ℝ) expressed as fractions with natural number numerators and denominators.

The very last theorem is a general fact about absolute values.

SML

```
val ℝ_frac_norm_conv : CONV;   val ℝ_frac_plus_frac_conv : CONV;
```

Description *ℝ_frac_norm_conv* proves theorems giving a normal form for real literals expressed as numeric fractions. The theorems have the form:

$\vdash m\ /_N\ n = \mathbb{N}\mathbb{R}\ m'$
$\vdash m\ /_N\ n = m'\ /_N\ n'$

where m, n, m' and n' are natural number literals and, in the second case, m' and n' are coprime and $n' > 1$.

ℝ_frac_plus_frac_conv proves theorems of the forms:

$\vdash m\ /_N\ n +\ m'\ /_N\ n' = t$

where m, n, m' and n' are natural number literals and t is in the same kind of normal form as produced by *ℝ_frac_norm_conv*.

Errors

116301 ?0 is not of the form ⌜$m\ /_N\ n$⌝ or ⌜$\mathbb{N}\mathbb{R}\ m$⌝
116302 cannot simplify ?0 because it denominator is 0.
116303 ?0 is already in normal form
116304 ?0 is not of the form ⌜$(i\ /_N\ m) +_R (j\ /_N\ n)$⌝

SML
```
val ℝ_greater_conv : CONV;        val ℝ_≥_conv : CONV;
val ℝ_subtract_conv : CONV;
```

Description These conversions prove theorems of the following forms:

$\vdash x > y \Leftrightarrow y < x$
$\vdash x \geq y \Leftrightarrow y \leq x$
$\vdash x - y \Leftrightarrow y + \sim x$

Errors
116313 ?0 is not of the form ⌜x > y⌝ where ⌜x⌝ and ⌜y⌝ have type ⌜:ℝ⌝
116314 ?0 is not of the form ⌜x ≥ y⌝ where ⌜x⌝ and ⌜y⌝ have type ⌜:ℝ⌝
116315 ?0 is not of the form ⌜x − y⌝ where ⌜x⌝ and ⌜y⌝ have type ⌜:ℝ⌝

SML
```
val ℝ_lin_arith_rule : TERM list -> THM;
val ℝ_lin_arith_tac : TACTIC;
val ℝ_lin_arith_prove_tac : THM list -> TACTIC;
```

Description Given a system, $\Gamma = [r_1, r_2, ...]$, of numeric constraints, r_i, of the form $(s_i{:}\mathbb{R}) = c_i$or $s_i \leq c_i$, where the c_i are rational literals, the rule attempts to prove a theorem of the form $\Gamma \vdash F$. Terms in Γ which are not of either of these forms are ignored and do not appear in the assumptions of the result theorem.

The tactics use the rule in much the same way as the integer linear arithmetics use ℤ_*lin_arith_rule* (see ℤ_*lin_arith_tac*). The usual interface to these functions is via the decision procedures in the proof context ℝ_*lin_arith*, q.v., e.g. as invoked by *PC_T1*"ℝ_*lin_arith*"*prove_tac*[]

The algorithm for the decision procedure is very similar to the one used in *lin_arith_rule*, q.v.

Errors
116341 The linear arithmetic proof procedure cannot prove this conjecture

SML
```
val ℝ_lin_qep_conv : CONV;
```

Description This conversion implements the Hodes-Fourier-Motzkin quantifier elimination procedure for linear arithmetic with real coefficients.

SML
```
val ℝ_max_cons_thm : THM;
val ℝ_max_conv_thm : THM;
val ℝ_min_cons_thm : THM;
val ℝ_min_conv_thm : THM;
```

Description These are the ML bindings for theorems in the theory ℝ about maxima and minima of lists.

SML
```
val ℝ_max_conv : CONV;
val ℝ_min_conv : CONV;
```

Description These conversions compute theorems of the form:

SML
```
val ℝ_minus_conv : CONV;
```

Description This conversion simplifies expressions involving real negation. It returns theorems of the form:

```
⊢ ~(~ x ) = x
⊢ ~(0/1) = 0.
⊢ ~0. = 0.
```

Errors
```
116319 ?0 is not of the form ⌜~ (~ x)⌝, ⌜~(0/1)⌝, or ⌜~0.⌝
```

SML
```
val ℝ_monoid_delta_dense_thm : THM;
val ℝ_monoid_dense_thm : THM;
val ℝ_subgroup_dense_thm : THM;
val ℝ_semigroup_dense_thm : THM;
val ℝ_add_hom_image_group_thm : THM;
val ℝ_add_hom_kernel_group_thm : THM;
```

Description These are the ML bindings for theorems giving various properties of substructures of the additive group ℝ and of (additive) homomorphisms from ℝ to ℝ.

SML
```
val ℝ_opah_thm : THM;               val ℝ_opah_strict_thm : THM;
val ℝ_opah_one_one_thm : THM; val ℝ_opah_dense_image_thm : THM;
val ℝ_opah_onto_thm : THM;          val ℝ_opah_inverse_add_hom_thm : THM;
val ℝ_opah_inverse_thm : THM;   val ℝ_opah_extension_thm1 : THM;
val ℝ_opah_extension_thm2 : THM;      val ℝ_opah_extension_thm : THM;
val ℝ_opah_order_thm : THM;     val ℝ_opah_eq_thm : THM;
val ℝ_opah_complete_thm : THM;
```

Description These are the ML bindings for theorems giving various properties of order-preserving additive homomorphisms from ℝ to ℝ.

SML
```
val ℝ_ord_pres_strict_thm : THM;
val ℝ_add_hom_0_thm : THM;
val ℝ_add_hom_minus_thm : THM;
val ℝ_add_hom_extension_thm : THM;
```

Description These are ML bindings for theorems about order-preserving functions and additive endomorphisms from ℝ to ℝ.

SML
```
val ℝ_over_times_recip_thm : THM;           val ℝ_times_recip_thm : THM;
val ℝ_eq_recip_thm : THM;               val ℝ_recip_clauses : THM;
val ℝ_0_less_0_less_recip_thm : THM;    val ℝ_over_cancel_thm : THM;
val ℝ_over_plus_over_thm : THM;             val ℝ_0_over_thm : THM;
val ℝ_over_1_thm : THM;                 val ℝ_¬_recip_0_thm : THM;
```

Description These are the ML bindings for some basic theorems about reciprocals (multiplicative inverses) in the real numbers and their relationship with the other structure.

SML
```
val ℝ_plus_assoc_thm : THM;            val ℝ_plus_comm_thm : THM;
val ℝ_plus_unit_thm : THM;       val ℝ_plus_mono_thm : THM;
val ℝ_plus_assoc_thm1 : THM;     val ℝ_plus_mono_thm1 : THM;
val ℝ_plus_mono_thm2 : THM;             val ℝ_plus_0_thm : THM;
val ℝ_0_1_thm : THM;                    val ℝ_plus_order_thm : THM;
val ℝ_plus_minus_thm : THM;      val ℝ_eq_thm : THM;
val ℝ_plus_clauses : THM;        val ℝ_minus_clauses : THM;
val ℝ_less_clauses : THM;        val ℝ_≤_clauses : THM;
val ℕℝ_plus_homomorphism_thm : THM;
val ℕℝ_plus_homomorphism_thm1 : THM;
val ℝ_minus_eq_thm : THM;               val ℕℝ_0_less_thm : THM;
val ℕℝ_one_one_thm : THM;               val ℝ_less_less_0_thm : THM;
val ℝ_less_0_less_thm : THM;            val ℝ_≤_≤_0_thm : THM;
val ℝ_≤_0_≤_thm : THM;           val ℕℝ_less_thm : THM;
val ℝ_less_strong_dense_thm : THM;val ℕℝ_≤_thm : THM;
val ℝ_sup_plus_thm : THM;               val ℝ_sup_plus_sup_thm : THM;
val ℝ_less_¬_eq_0_thm : THM;
```

Description These are the ML bindings for basic theorems about the additive structure of the real numbers and its relationship with the ordering relations and the supremum function.

SML
```
val ℝ_plus_conv : CONV;          val ℝ_times_conv : CONV;
val ℝ_less_conv : CONV;
val ℝ_≤_conv : CONV;                    val ℝ_eq_conv : CONV;
val ℝ_over_conv : CONV;          val ℝ_ℕ_exp_conv : CONV;
```

Description These conversions compute theorems of the form

$\vdash x\ op\ y = z$

where x and y are rational literals and op is the operation indicated by the name of the conversion. A rational literal is taken to be an optionally signed expression made up from natural number literals using ℕℝ or $/_N$. (Note that ℕℝ applied to a natural number literal is pretty-printed and may be entered as the natural number followed by a decimal point. The term z is a truth value or an optionally signed rational literal in normal form (see ℝ_*frac_norm_conv* as appropriate.

Errors

116305 ?0 is not of the form $\ulcorner x +_R y \urcorner$ *where* $\ulcorner x \urcorner$ *and* $\ulcorner y \urcorner$ *are rational literals*
116306 ?0 is not of the form $\ulcorner x *_R y \urcorner$ *where* $\ulcorner x \urcorner$ *and* $\ulcorner y \urcorner$ *are rational literals*
116309 ?0 is not of the form $\ulcorner x < y \urcorner$ *where* $\ulcorner x \urcorner$ *and* $\ulcorner y \urcorner$ *are rational literals*
116310 ?0 is not of the form $\ulcorner x \leq y \urcorner$ *where* $\ulcorner x \urcorner$ *and* $\ulcorner y \urcorner$ *are rational literals*
116311 ?0 is not of the form $\ulcorner x = y \urcorner$ *where* $\ulcorner x \urcorner$ *and* $\ulcorner y \urcorner$ *are rational literals*
116312 ?0 is not of the form $\ulcorner x\ /\ y \urcorner$ *where* $\ulcorner y \urcorner$ *is a rational literal*
116317 ?0 is not of the form $\ulcorner x \ \hat{}_N\ m \urcorner$ *where* $\ulcorner x \urcorner$ *is a rational number literal and* $\ulcorner m \urcorner$ *is a natural number literal*

SML

```
val ℝ_plus_def : THM;            val ℝ_subtract_def : THM;
val ℝ_≤_def : THM;         val ℝ_less_def : THM;
val ℝ_greater_def : THM; val ℝ_≥_def : THM;
val ℝ_minus_def : THM;   val ℝ_times_def : THM;
val ℝ_over_def : THM;            val ℝ_frac_def : THM;
val ℕℝ_def : THM;          val ℝ_ℕ_exp_def : THM;
val ℝ_abs_def : THM;             val ℤℝ_def : THM;
val ℝ_ℤ_exp_def : THM;           val float_def : THM;
val ℝ_max_def : THM;             val ℝ_min_def : THM;
```

Description These are the ML bindings for the definitions in the theory ℝ.

SML

```
val ℝ_sup_thm : THM;             val ℝ_less_sup_thm : THM;
val ℝ_less_sup_bc_thm : THM;     val ℝ_≤_sup_thm : THM;
val ℝ_≤_sup_bc_thm : THM;        val ℝ_∈_≤_sup_bc_thm : THM;
val ℝ_⊆_sup_thm : THM;   val ℝ_sup_≤_bc_thm : THM;
val ℝ_sup_less_bc_thm : THM;     val ℝ_sup_eq_bc_thm : THM;
val ℝ_eq_sup_bc_thm : THM;       val ℝ_less_sup_∈_thm : THM;
```

Description These are the ML bindings for theorems in the theory ℝ to assist with reasoning about the values of the supremum function.

SML

```
        val ℝ_term_order_comb : TERM ORDER -> TERM ORDER;
        val ℝ_term_order_comb1 :
                {graded : bool, inverse: bool} ->
                TERM ORDER -> TERM ORDER;
```

Description These are the term order combinators used as arguments to *make_term_order* to implement ℝ_*term_order* and ℝ_*term_order1*, q.v.

SML
```
val ℝ_term_order : TERM ORDER;
val ℝ_term_order1 : {graded : bool, inverse: bool} -> TERM ORDER;
```

Description ℝ_*term_order* gives an ordering relation on HOL terms analogous to that given by *term_order*, q.v., but which takes special arrangements for certain terms of type ℝ. Terms of the form $t_1 * ... t_l$ where each t_i are called monomials and are ordered lexicographically using an ordering on the t_i which takes terms formed by exponentiation by natural number literals in the following order: $x\hat{}\ 0 < x < x\ \hat{}\ 1 < x\ \hat{}\ 2 < ....$

Rational literals, i.e., terms formed using ℕℝ or natural number division with natural number literal operands are ordered numerically.

Each monomial is grouped next to the the same monomial with coefficients and signs or reciprocals (where a coefficient is a rational literal appearing in a monomial as the term t_1. Thus:

```
a*b < ~a * b < 2. * a * b < ~(2/1)^-1 * a * b < ....
```

Note that ℝ_*term_order*takes the lexicographic order in the sense appropriate to a sparse representation of monomials. This means that xy precedes xz (whereas in a dense representation, $xz = x^1y^0z^1$ precedes $xy = x^1y^1z^0$). This is intuitive, but does not give an admissible order in the sense of Gröbner basis theory, since, for example, xy is collated before xz, but $xy^2 = (xy)y$ is collated after $xyz = (xz)y$.

ℝ_*term_order1* provides the usual admissible lexicographic orderings used in Gröbner basis theory (following section 5.1 of T. Becker and V. Weispfenning, *Gröbner Bases*, Springer, 1993). For example, ℝ_*term_order1*{*graded* = *true*, *inverse* = *true*} gives the graded inverse lexicographic order.

See Also ℝ_*term_order_comb*, *graded*_ℝ_*term_order_comb*

SML
```
val ℝ_times_assoc_thm : THM;              val ℝ_times_unit_thm : THM;
val ℝ_times_comm_thm : THM;               val ℝ_times_assoc_thm1 : THM;
val ℝ_times_plus_distrib_thm : THM;
val ℝ_0_less_0_less_times_thm : THM;      val ℝ_times_order_thm : THM;
val ℝ_times_clauses : THM;
val ℝ_times_0_thm : THM;                  val ℝ_times_1_thm : THM;
val ℝ_times_cancel_thm : THM;             val ℝ_times_eq_0_thm : THM;
val ℝ_times_minus_thm : THM;
val ℕℝ_times_homomorphism_thm : THM;
val ℕℝ_times_homomorphism_thm1 : THM;
val ℝ_times_mono_⇔_thm : THM;             val ℝ_times_mono_thm : THM;
val ℝ_0_≤_0_≤_times_thm : THM;            val ℝ_cross_mult_less_thm : THM;
val ℝ_cross_mult_eq_thm : THM;
```

Description These are the ML bindings for some basic theorems about the multiplicative structure of the real numbers and its relationship with the additive and order structure.

SML

```
val ℝ_unbounded_below_thm : THM;
val ℝ_unbounded_above_thm : THM;
val ℝ_less_irrefl_thm : THM;        val ℝ_less_antisym_thm : THM;
val ℝ_less_trans_thm : THM;         val ℝ_less_cases_thm : THM;
val ℝ_≤_cases_thm : THM;            val ℝ_≤_less_cases_thm : THM;
val ℝ_eq_≤_thm : THM;               val ℝ_less_≤_trans_thm : THM;
val ℝ_≤_less_trans_thm : THM;val ℝ_≤_refl_thm : THM;
val ℝ_≤_antisym_thm : THM;          val ℝ_≤_trans_thm : THM;
val ℝ_≤_¬_less_thm : THM;           val ℝ_¬_≤_less_thm : THM;
val ℝ_less_¬_eq_thm : THM;          val ℝ_¬_less_≤_thm : THM;
val ℝ_less_dense_thm : THM;         val ℝ_complete_thm : THM;
```

Description These are the ML bindings for theorems in the theory ℝ concerned with basic properties of the ordering relations. Note that $>$ and $\geq$ are defined in terms of $<$ and $\leq$ and the latter are the preferred form in all theorems in this theory.

SML

```
val ℝ_ℤ_exp_conv : CONV;
```

Description This conversions computes theorems of the form and

$$\vdash x \mathbin{\hat{}_Z} \mathbb{NZ}\ m = x \mathbin{\hat{}_N} m$$
$$\vdash x \mathbin{\hat{}_Z} \sim(\mathbb{NZ}\ m) = (x \mathbin{\hat{}_N} m)^{-1}$$

Errors

116321?0 is not of the form ⌜$x \mathbin{\hat{}_Z} m$⌝

SML

```
val ℤℝ_plus_homomorphism_thm : THM;
val ℤℝ_minus_thm : THM;
val ℤℝ_ℕℤ_thm : THM;
val ℤℝ_times_homomorphism_thm : THM;
```

Description These are the ML bindings for theorems in the theory ℝ to assist with reasoning about the embedding of the integers in the reals.

REFERENCES

[1] Michael J.C. Gordon and Tom F. Melham, editors. *Introduction to HOL.* Cambridge University Press, 1993.

[2] Michael J.C. Gordon, Arthur J. Milner, and Christopher P. Wadsworth. *Edinburgh LCF. Lecture Notes in Computer Science. Vol. 78.* Springer-Verlag, 1979.

[3] DS/FMU/IED/USR001. *ProofPower Document Preparation.* Lemma 1 Ltd.

[4] DS/FMU/IED/USR004. *ProofPower Tutorial Manual.* Lemma 1 Ltd., `http://www.lemma-one.com`.

[5] DS/FMU/IED/USR007. *ProofPower Installation and Operation.* Lemma 1 Ltd., `http://www.lemma-one.com`.

INDEX

KEYWORD INDEX